

<i>Section I Report</i>	<i>2</i>
<i>Chapter 1 Introduction</i>	<i>3</i>
1 Study of the existing technologies and projects.	4
2 Motivation.....	5
3 Objectives.....	6
4 Methodology.....	6
5 Preview Considerations.....	7
<i>Chapter 2 Hardware</i>	<i>8</i>
<i>Chapter 3 Software.....</i>	<i>26</i>
Algorithms	27
1. Evasive Robot	29
2. Easy Avoidance	48
3. From A to B: Straight	51
4. AB: The least X / The least Y	56
5. Line Follower	60
6. Hunter	68
7. Tremaux Maze Solver	72
8. Interface for Robot Patrol System	83
<i>Chapter 4 Conclusions</i>	<i>88</i>
<i>Capítulo 5 Future developments</i>	<i>90</i>
<i>Bibliography.....</i>	<i>92</i>
<i>Section II User Manual.....</i>	<i>93</i>
<i>Section III Source Code.....</i>	<i>104</i>
<i>Section IV Datasheets</i>	<i>202</i>

SECTION I REPORT



CHAPTER 1 INTRODUCTION

In this chapter it exposes a brief explanation about the project carried out, which deals with developing a system of navigation for a robot.

As one of the characteristics and conditions in the execution of this project is to program the navigation system in JAVA, thus the first step implies to invest a certain and considerable time in learning this programming language and consequently to be introduced in the world of Object Oriented Programming for first time.

After making a study about the structural possibilities, it continues with the analysis of the options to manage the environments and chosen challenges. Subsequently it develops the appropriate algorithms taking into account the alternatives and the technology which it is working.

In the next sections it talks about the technology which it is going to work with and it mentions the motivation and objectives of the project, and some preview considerations.



1 Study of the existing technologies and projects.

As for the robot navigation, this is the first contact with this aspect of field of robotics, which is a very important subject as it is requested in so many applications, because independently of the function or objective of any robot, it must know to cope with the environment in which is carrying out its tasks.

But regarding the technology is being used in this project is from LEGO MINDSTORMS, specifically the new so called NXT technology, developed by this company, directed to improve the characteristics and possibilities of its products for the fans of robotics.

It is not difficult to find out, by surfing a little the internet, that these robots from LEGO are not only a enjoying entertainment as technological hobby but also a really useful and versatile educational tool. The show is a great quantity of schools, high schools and universities (that is, at all levels) that use this kind of tool (included, as could have been guessed, the lab in which this project has been developed) to teach some technologic courses to the students. That is why this project has been executed with this tool.



2 Motivation

In the personal perspective as in the technological view, this Project is considered very interesting. Actually any task related to robotics underlies a look to future. Some of the algorithms answer to standard robot behavior for competitions, so in this case is more like a leaning and put in practice than a selling purpose; and other ones are focus on a navigation a little closer to reality, so having many kind of applications.

In the commercial sense, a robot provided with a navigation system, enabled to avoid obstacles and reach the target place could be requested for many tasks. For example, a robot, which is working in a department store, in charge of providing help to customers, is called for somebody from anywhere. Then the robot has to go to that place where its help is requested, by avoiding walls, furniture, people...and it has to know the direction to get there.

As the example explained above, many other applications are possible for a robot with these characteristics.



3 Objectives

In general terms it can say clearly that the main objective is to provide a robot with a navigation system composed of some different algorithms for enabling the robot to manage the diverse environments which, of course, are determined in advance.

In a little more detail, the whole of algorithms can be classified in two groups: Those are focus on any standard competition such as Maze Solver, LineFollower or Hunter-Prey; and those are not focus on any standard, such as Obstacle Avoidance. In the case of the last one, anyway there will have to be some restrictions in the environment for a suitable accomplishment of the goals.

4 Methodology

The way to get the eventual objective is a logic process which consists of studying the object oriented programming language called JAVA at first time. Later, it has to study the workspace which it is going to work with, called Eclipse (a useful tool supporting JAVA). After that it carries out the main stage, that is the development of the algorithms, by considering the alternatives, requirements and capabilities.

Finally the robot is built with a Lego Brick, some environments are made if necessary and the algorithms are put in practice.



5 Preview Considerations.

Some matters must be mentioned at first:

There are some critical restrictions coming from the hardware, that is, from the Lego robot. Despite being a good educational and entertainment tool, we cannot forget that after all it is a toy.

That means that the answers of the robot many times are not optimal at all, showing unfortunately sometimes a poor behavior, specially the sensors, which has a considerable lack of precision. Other restrictions are that only can be connected four sensors to the control and the available tool is a Lego Basic Brick, so the structural options are limited in reality.

This problem will be present during the project and it will be mentioned when necessary to underline.

That is why in this sense the different algorithms can now be divided in other two wholes: Those are too sophisticated for the Lego robot and consequently to put in practice, and those are capable or simplified to be embedded in the robot.



CHAPTER 2 HARDWARE

In this chapter it is going to explain and analyze the possibilities that the Lego tool offers. It must understand the physic support where the software will be applied and the limitations, in order to find out easily the potential and practical troubles, and to solve them; or even to look for better structural features to simplify the necessary application.

Most robots are designed with some kind of mobility in mind. Motion makes this machines animated. Most mobile robots belong to one of two categories: *wheeled* robots or *legged* robots. Though legs provide an effective way to move on rough terrains, wheels are generally much more efficient on smooth surfaces. Attention will be focus on surveying some common wheeled mobility configurations, discussing some of their pros and cons.



Simple Differential Drive

This architecture has many advantages, especially in its simplicity, that is why it is the most often used configuration for LEGO mobile robots.

A differential drive is made of two parallel drive wheels on either side of the robot, powered separately, with one or more casters (pivoting wheels) which help support the weight but that have no active role because the robot motion vector results from two independent components.

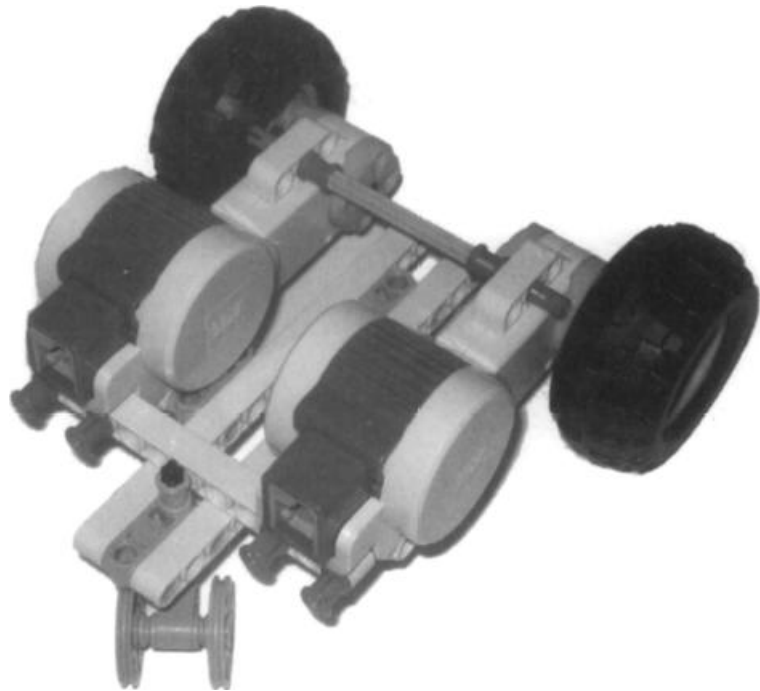


Figure 1: Simple Differential Drive Ref {DAMG07}





Figure 2: Real built Differential Drive

In the case of the two drive wheels turn in the same direction at the same speed, the robot goes straight. But when the the wheels rotate at the same speed but in opposite directions, the robot turns in place, pivoting around the midpoint of the line that connects the drive wheels. Table 1 shows the behavior of a differential drive robot respect to the direction of its wheels.



Left Wheel	Right Wheel	Robot
Stationary	Stationary	Rests stationary
Stationary	Forward	Turns counterclockwise, pivoting around the left wheel
Stationary	Backward	Turns clockwise, pivoting around the left wheel
Forward	Stationary	Turns clockwise, pivoting around the right wheel
Forward	Forward	Goes forward
Forward	Backward	Spins clockwise in place
Backward	Stationary	Turns counterclockwise, pivoting around the right wheel
Backward	Forward	Spins counterclockwise in place
Backward	Backward	Goes backward

Table 1: Differential Drive behavior respect to the wheel direction
Ref {DAMG07}

At different combinations of speed and direction, the robot makes turns of any possible radius. This maneuverability, the capability to turn in place in particular, makes the differential drive the ideal candidate for many kind of projects and applications. And taking into account that it is very easy to implement, that is why a significant percentage of all mobile LEGO robots belong to this category.

It is really useful for *tracking* the robot position, and it requires very simple math.

Only one main drawback is found out, It is not easy to get the robot to move in a perfectly straight line. Because the motors have a different efficiency, surely it may find one wheel turning a little faster than the other, thus making the robot turn slightly left or right. In some



projects, this isn't a problem, particularly those programmed for continuous route correction, such as following a line. But when it wants the robot to simply go straight in an open space, this can be a important problem.

This relevant problem may be solved by embedding encoders in the motors (included in the NXT technology). In a nutshell, the encoders allow motor rotations to be monitored to the nearest degree. This provides the way for the robot to go straight and auto-correct themselves along the way.

Despite this new feature, it is still important to understand the concepts behind differential drive mechanisms. Later it is discussed alternative approaches including both the sensor-based approach as well as using gears.

Using Servo Motor Encoders to Go Straight

A more sophisticated approach that has several positive side effects requires a feedback mechanism into the system, thus controlling each wheel with sensors and adjusting their speed according to the readings. This is the most common way for real differential drives. The only requirement is that it needs two drive motors, each attached to a drive wheel.

Using Gears to Go Straight



It functions in much the same way as a vehicle differential operates, the basic principle being that the motor drives the differential housing via a main drive gear, "A". The differential is geared such that both wheels turn at the same direction and speed, "B" and "C". This enables smooth transfer of power when a vehicle is turning. Both NXT motors drive each wheel directly, and are connected to the differential unit via a series of gears such that their directions are inverted. When both motors are driving the same speed, the differential unit will remain stationary. If the motors are not in sync, the differential will turn for the duration that the motors are not in sync. it provides a means to monitor when motors are driving at different speeds. When they are driving in sync, the differential will remain stationary.

Using Casters to Go Straight

Casters are another key factor in getting the differential drive moving and turning smoothly. Usually, they are not given enough consideration. Unfortunately, its design is limiting and it will skid or jam. It uses two wheels coupled on the same axle and doesn't allow the wheels to turn independently.

In the assembly on the left, the axle turns with the wheel, whereas the one on the right has the wheels spinning on the axle.





Figure 3: Coupled caster Ref {DAMG07}



Figure 4: Caster to avoid skidding Ref {DAMG07}



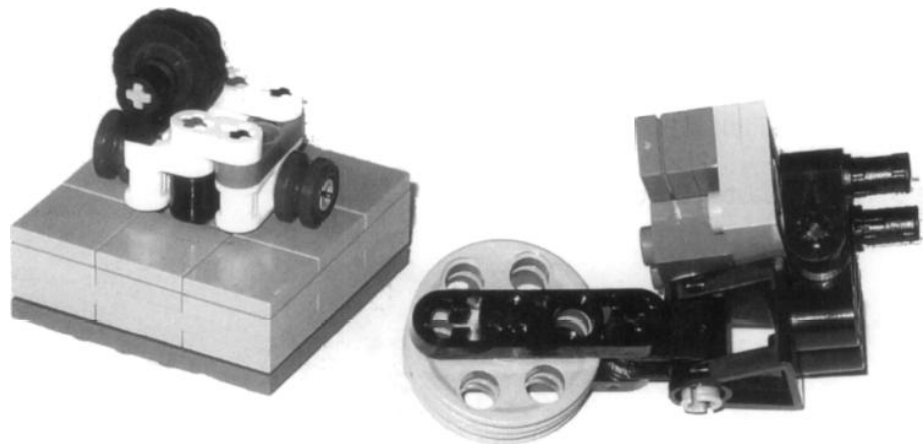


Figure 5: Other casters Ref {DAMG07}

Selecting one or several casters depends on the specific work or task for which the robot is being designed. For example, a single caster is enough for most applications, including which are approached in this project, but two casters at the front and rear of the robot are a better option when stability is important, being a characteristic at risk.

Another option, not so known or common, it is to use a plastic ball, instead of casters. If the robot is moving on a smooth surface, a large ball performs like as a caster.

. The advantage of this approach is that it has not caster wheels to get caught up, and the ball will turn if it can. This approach will not work well if the robot is moving across a surface that has a rubbery texture.



Although the ball turns freely within the cup structure, it is still subject to friction within it.

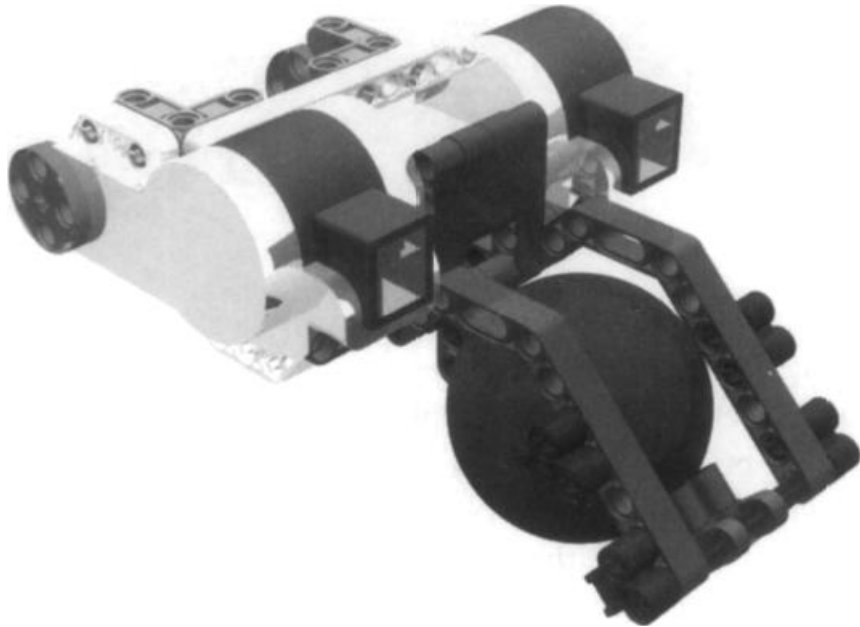


Figure 6: Differential Drive with plastic ball implemented.

Skid-Steer Drive

This is a variation of the differential drive. Usually it is used with tracked vehicles, but sometimes with four- or six-wheel platforms as well. In the case of tracked vehicles, this architecture is the only driving option. There are in reality quite good examples of skid-steer drives, such as excavators, tanks. Employing an additional set of tracks at the front, it can pivot to provide additional navigation functionality. Each track is powered by its independent motor. The advantage of a skid-steer drive is its capability to turn on the spot, which allows vehicles of this



type to operate in tight areas. The downside to this is that when it is turning it appears a considerable quantity of friction on surfaces such as carpets. If you are running rubber tracks, in that sense it also get a extra coefficient for climbing capability.

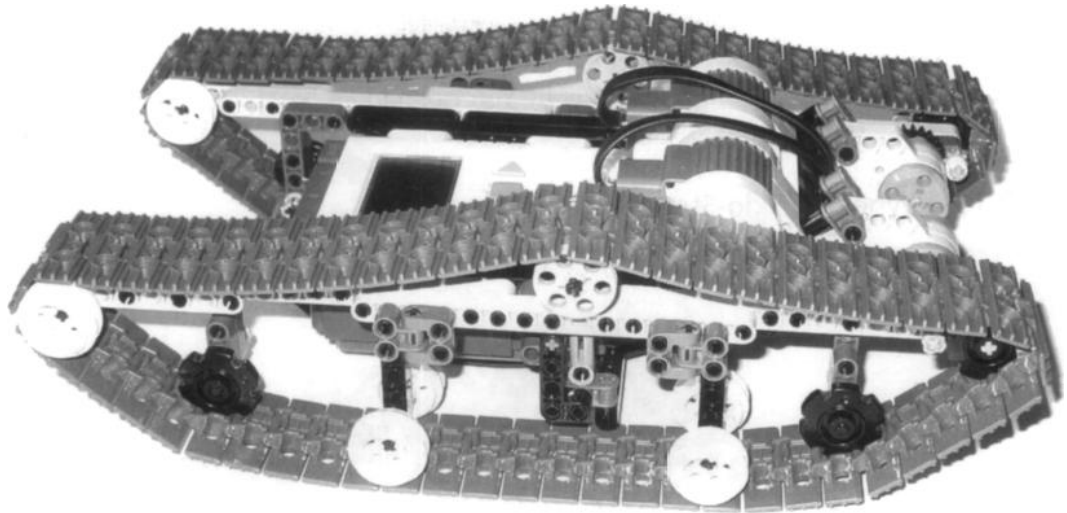


Figure 7: A tracked Skid-Steer Drive Ref {DAMG07}

In a wheeled skid-steer drive may be necessary to include some characteristic: It should transmit the power to all the wheels, because otherwise, probably it will not turn softly, or even anything.

In the next example, it is connected by a common chain drive for each side receiving power from two motors, such as is shown in the tracked style. This platform performs with quite strength and it is also fast, compact, and even light.





Figure 8: Wheeled Skid-Steer Drive Ref {DAMG07}

Skid-steer drives have some common characteristics with differential drives, for example, they both have the same difficulties in going exactly straight.

However, this architecture has its own features:

Tracks moving on rough floors have a better grip than wheels, but not on soft ones.

Tracks implies quite friction that spend some of the power supplied by the motors.

Because of the unavoidable skidding in this architecture, it has not chosen this option, as it is very disadvantageous for applications in which it must calculate the position of the robot by its position.



Steering Drive

A steering drive is the most typical style which is used in many vehicles and cars. It is characterized by two front steering wheels and two fixed rear wheels. The best way usually is to conduct the rear one.

However, this is less versatile than differential drives, and impossible to steer in place. Among the advantages of this configuration it can find:

Easy capability to drive straightly and quite stability

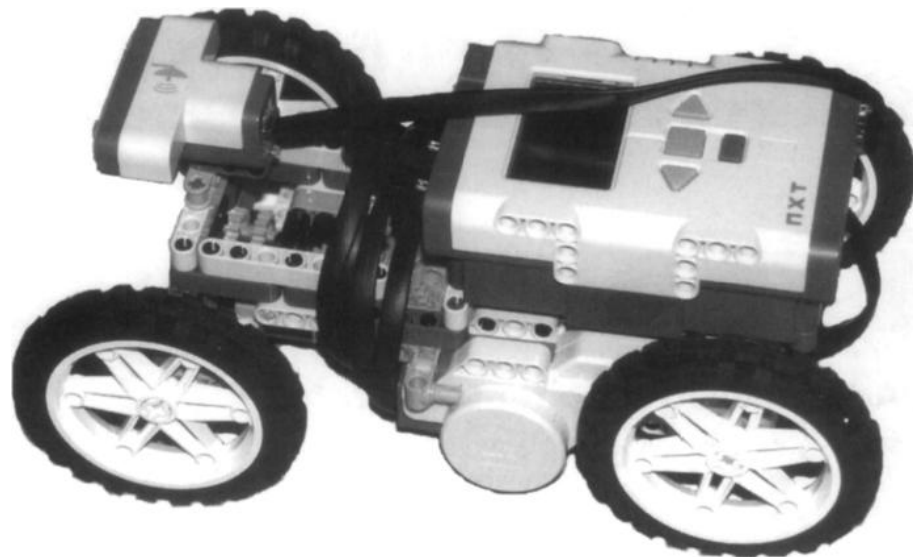


Figure 9: Steering Drive Ref {DAMG07}

Two motors drive the rear wheels (one for each) and one motor drives the steering mechanism. It implies a different control of steering and driving motors, consequently allowing more precision in steering, but



deficient capability to turn in place.

In some cases it could be necessary to turn the robot smoothly, and in the usual conditions about wheels and floor might become difficult, as in reality when the robot or vehicle turns, the inner wheel goes along a tighter bend than the outer one. In large radius turns the difference is not appreciable, but in small radius turns become relevant, and that causes one of the wheels skids. Then it can apply the Ackerman Steering to avoid the skidding and consequently the difference of angle among wheels. It works on the next way: When the lines extended from any wheel axle meet and revolve around one common point, the vehicle turns smoothly. It is shown in the next figure.

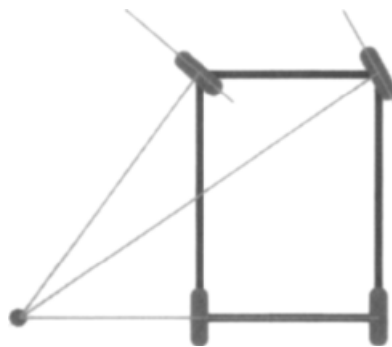


Figure 10: Ackerman Steering scheme Ref {DAMG07}

It can see a wheel allocated just below the pivoting axle, which does not affect the steering. If the wheel is allocated behind its steering column, friction causes the dynamic forward motion of the car to push the wheels toward the rear, it implies a self-centering action. A good



example is the well known of a shopping cart, it can appreciate that the current wheel contact area is behind the pivoting axis. The more the wheel is moved behind the pivoting axis, the more self-centering it gets.

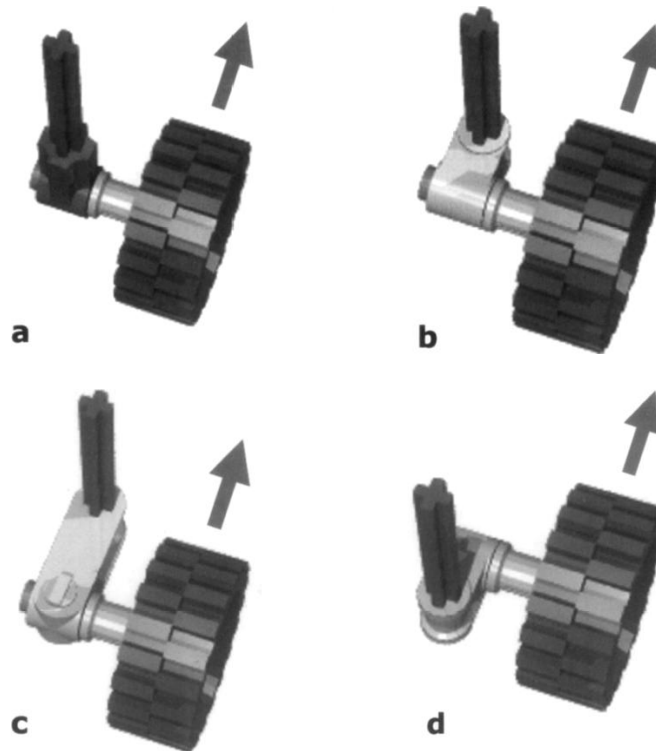


Figure 11: Movement of wheel from pivoting axle Ref {DAMG07}

The steering drive is a suitable configuration for rough terrains, because it is very stable on its four wheels. It is possible to improve the grip with the ground if any kind of suspension is used. An important point to take into account is that none of the drive wheels became in any time without contact with the ground; otherwise, the differential would find the way of least resistance and consequently it would transfer all the power to that wheel, and finally the result would be in the wheel spinning and the robot getting immobilized.



If the wheel axle is connected to an extra common axle with a elements such pulleys and belts, a limited slip differential would help reduce this problem. Usually the belts keep the driven axles rotating at the same velocity; however during turns they slip a little on their pulleys, and so allowing the wheel to adjust their speeds. If a wheel get lost the contact with the floor, the belts will still be able to transfer a part of power to the other wheel.

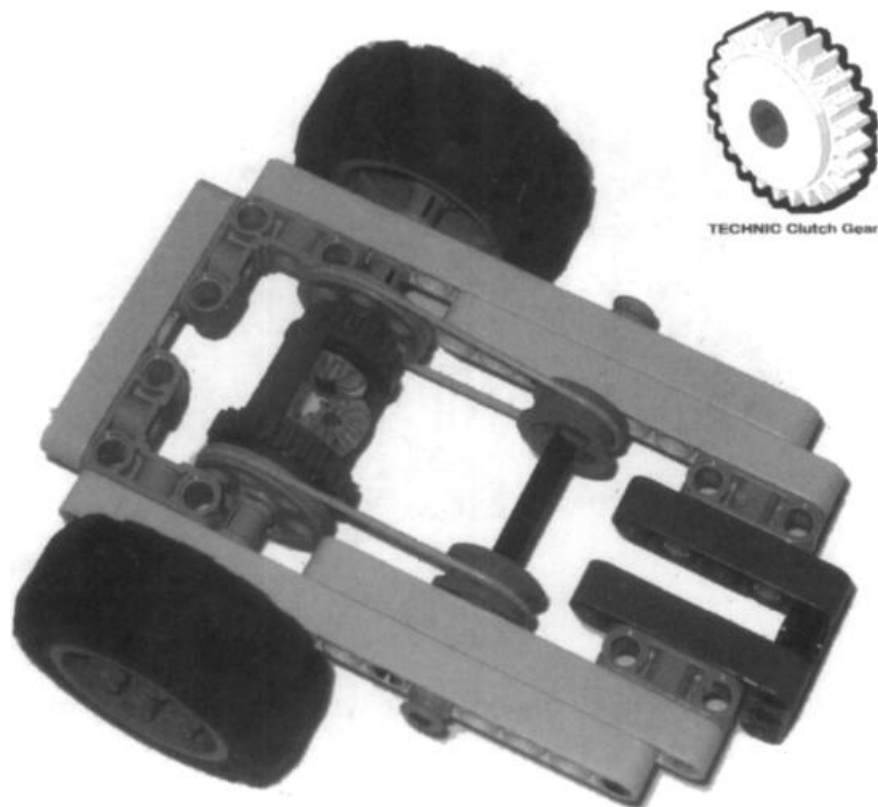


Figure 12: A limited slip differential Ref {DAMG07}



Synchro Drive

A synchro drive uses three or more wheels, all of them driven and steering. All wheels turn together in synchronization, always remaining parallel; thus, the robot changes its direction of movement without having to change its orientation.

The most important thing in order to make a 360-degree synchro drive and be able to avoid all limitations in its capacity to turn is to transfer motion along the pivoting axle of each wheel. The easiest approach implies a special part called the *turntable*, a large, round, rotating platform usually used in LEGO models. It can attach two pulley wheels and drive it with an axle that passes through the center of the turntable. This is necessary because the wheel must be connected to the part of the turntable that gets rotated by the external gear.

To build a complete synchro drive, it is necessary at least three of these turntables. Then it has to connect them in the way of that one motor can drive all the axles at the same time, while another can turn all the wheels in synchronization.



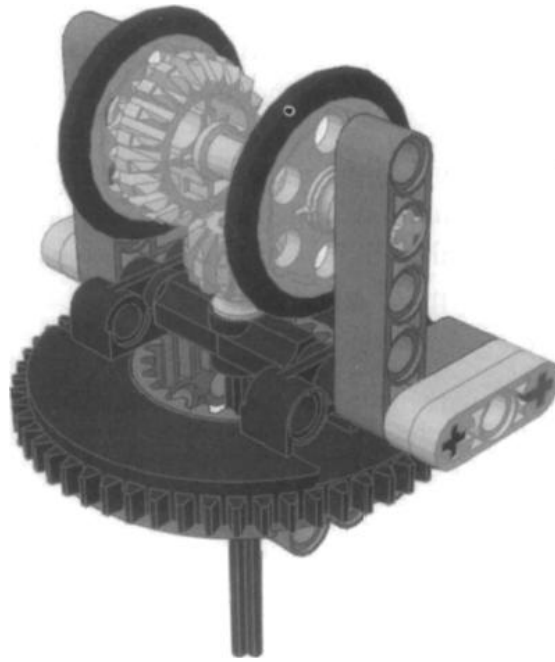


Figure 13: An assembled wheel for Synchro Drive

Other Configurations

Some more sophisticated or specialized possibilities for mobile architectures.

Multi-Degree-of-Freedom (MDOF): This option has three or more wheels, or even groups of them, both independently turned and driven. It has to guess a synchro drive in which each wheel can change its velocity and orientation without connection to the others: A robot with this characteristic would be able to behave like a differential drive, a steering drive, or a synchro drive only by controlling its configuration from the software. They are really difficult to build and control.



Articulated drive: It is quite similar to the steering drive, but instead of steering the wheels, it steers a complete section of the robot. The front wheels always remain parallel to the front part of the chassis, and the same applies to the rear wheels in regard to the rear part of the chassis. However, the two parts connect through a point of articulation that lets them pivot in the middle. This architecture is usual in wheeled excavators and other construction equipment..

Tri-Star wheel drive: This option has been designed for high capacity of mobility. Each "wheel" is in reality an equilateral triangle with wheels in each vertex; the robot features three of them for a total of 12 wheels. The wheels turn, and the triangles can also turn like larger wheels. During usual movements, two wheels of each triangle touch the ground, but when a wheel sticks against an obstacle, a complex gearing system transfers motion to the triangular structure, which turns and places its upper wheel past the obstacle.

After having studied all these interesting structural options for building the robot and taking into account the limitation of resources and the characteristics and types of applications it is going to face, it conclude that the best option (with a generalist and versatile view) is the Simple Differential Drive. So it is which has been brought to practice.



CHAPTER 3 SOFTWARE

In this chapter two different parts are involved in the software subject of this project. On the one hand it is about the informatics tool used to program the applications and tasks developed, and the alternatives which it could have also worked with. In this case it has considered several options: Eclipse, Lego Mindstorm, Matlab and Labview. On the other hand, it explains the developed algorithms their-selves and the diverse possibilities conceived. In this sense, the algorithms can be classified, as already mentioned before, in two groups, those are too sophisticated to put in practice into the Lego robot and those simpler and suitable for practice and trials.

Always it keeps clear that all process must be carried out in JAVA object oriented programming language, as one of the preview conditions of this project.

Actually, regarding the possible tools, the only true option is Eclipse workspace, as the project must be developed in JAVA. So in the section of User Manual a installation guide is included.



ALGORITHMS

In this subsection it explains the algorithms that have been conceived and programmed. Firstly it has approached some methods for obstacle avoidance, regarded as the main goal in the project development and divided in according to complexity or resources availability (as it can be sensors). Among them, such as Evasive Robot, tracking from A point to B point, the chosen one (Easy Avoidance Robot) to be implemented in the robot will be the background for other algorithms which take advantage of this basis. In this case of avoiding obstacles, the environment is quite simple, just the obstacles their-selves. However, the suspected too simplicity is due to the restrictions of the obstacles; it is detailed later.

Following with the same background it has developed a trial of Hunter behavior game, which it has not been put in practice because of lack of resources and other difficulties.

Later, a couple of Line Followers were developed and tried with excellent results. Although might be a quite simple target to accomplish, it considered mandatory to carry it out, as it is a fundamental application for any robotics fan. Moreover it is regarded as other kind or environment where the robot has know how to manage successfully fulfill its objective.

The next step was to program a method for maze application, where the robot has to find the way out in a determined labyrinth. It is



another typical example of application for mobile robotics, but in this case it has made a different approach.

Finally, it decided to go a little beyond the original purpose by adding a complementary tool for this whole patrol system. For that it developed a program with which it can control the robot from the computer with a interface, by Bluetooth communication.



Evasive Robot (Evasivebot)

This is the first algorithm programmed and it is also the most sophisticated, so it could not be tried because of the limitations of Lego robot, but here it is going to be explain because it is considered interesting enough and it would be possible to implement in a higher quality robot more suitable and versatile.

It can say that this algorithm try to get the minimum distance between two points whose trajectory is covered by a series of obstacles. And as the robot never knows the environment in advance, at first it will try to get straight by surrounding the appeared obstacles on the way of following the straight trajectory after avoidance.

As for the obstacles, it has to mention that there are some restrictions in order to be the robot successful in its behavior. All obstacles used in this application must be squared-like, that is, having right angles, such as squares, rectangles and similar ones in different longitudes and sizes.



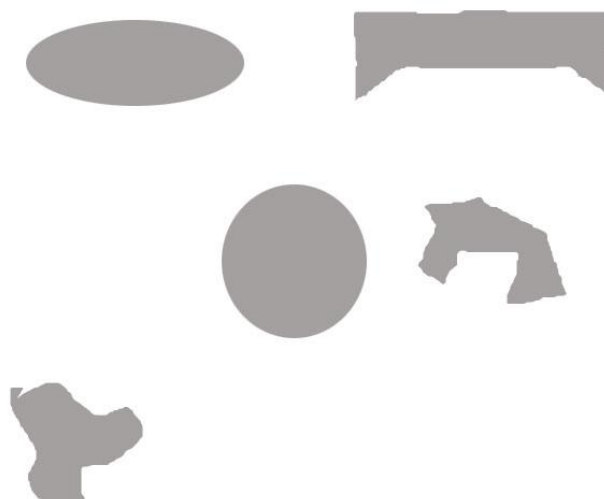


Figure 14: Variable-shaped obstacles

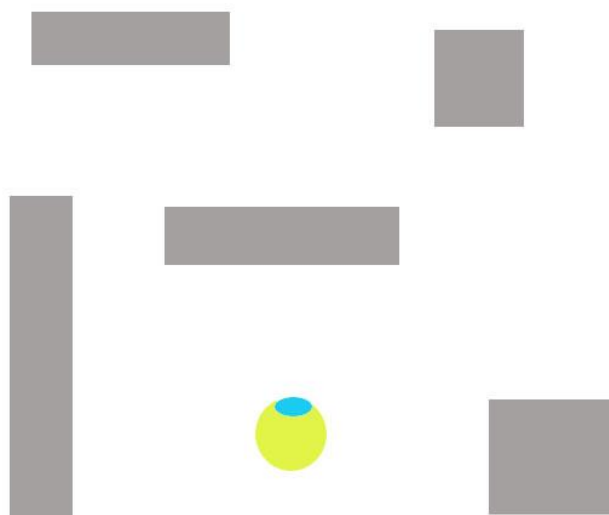


Figure 15: Squared Obstacles Avoidance



And it is not the only important restriction, because another basic factor is the obstacle has to be perfectly pulled out in front of the facing robot. That means, perpendicular to the current robot trajectory. If the obstacle is not right angled to the robot sensor shoot, the obstacle probably will not be detected. The usual perform results in ray deviation from the sensor, thus the sensor receiver does not get the signal implying there is an obstacle near the current position.

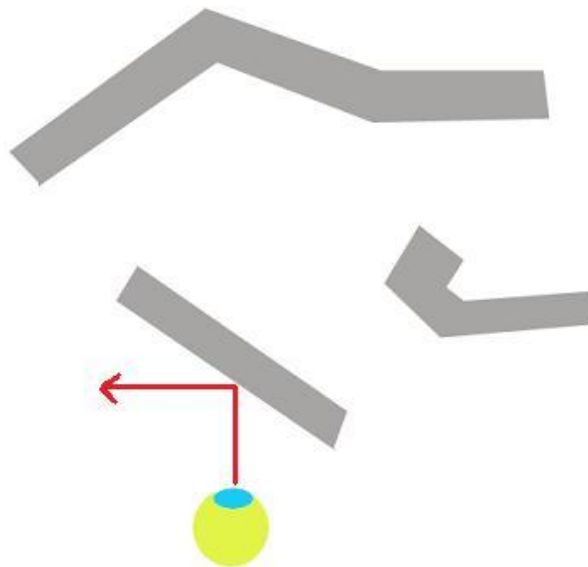


Figure 16: Facing a non right angled obstacle

All this limitations in the environment features are mainly due to that the robot use an ultrasonic sensor, so its rays must be straightly



directed and returned. Otherwise, the detection is never successful.

Also it takes into consideration that in this first algorithm the robot just can be provided with an only sensor.

Moreover, in this case, the angular range of the sensor is theoretically unknown, so it has only got a approximation from the real trials, with the consequence of the variability in the measurements, being less reliable ones. Under these circumstances, it would be appropriate to take a possible approximation of 30° , that means, 15° spread in each side from the straightforward robot vision line. This supposition in many cases does not underlay great difference.

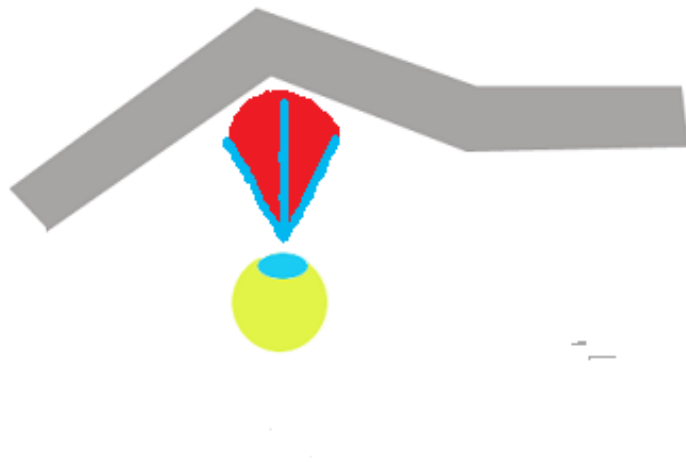


Figure 17: Robot sensor radial range

Before facing the obstacle, the robot looks around and calculates which the shortest way is to surround the obstacle. For that, the robot turns covering a middle circumference; take a measure from both



extremes of the obstacle, by performing the frontal sensor.

So far it is the beautiful theory, but actually this technique cannot work with no extra additions. Just in exceptional situations as facing a corner or similar, the rays being shot would become returned to the correct receiver position, but usually this will not happen, as trying to measure the distance to the extreme of the obstacle, the rays would go away and finally lost useless.

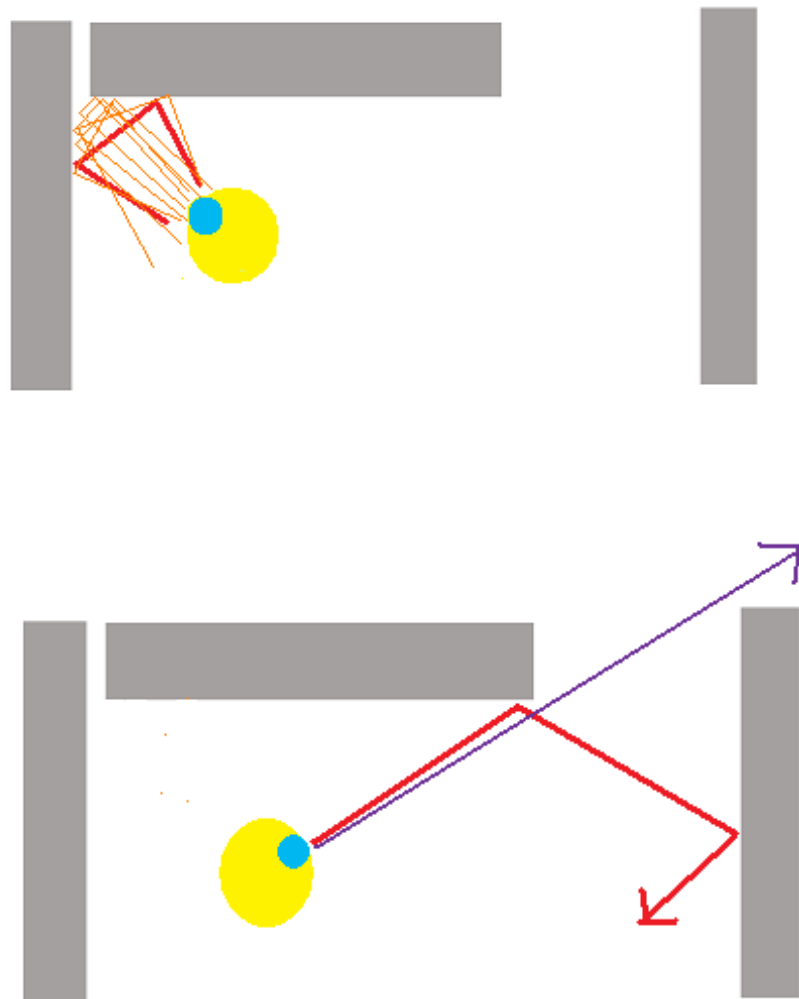


Figure 18: Shortcut calculation



The idea implemented in the source code is to detect the cornered end of the obstacle by making sensor shots until the signal is not more received, that means the obstacle is finished and then some calculations are made to register the resulting distance.

The conclusion is that it needs to receive the signal from the obstacle bound for working with this technique. For that it must do some change. There are several options to consider. Just one of them is referred to the robot: If it used a commercial ultrasonic sensor featured by burst-type and wider radial range, then it exists more possibilities to obtain the signal if the robot is close enough to the obstacle bound. But as it has to manage with Lego robot, this is not a feasible possibility, so it focuses the attention on the other alternatives which are based on the environment characteristics:

The first environmental change possible is to modify the obstacle's corner. It must transform the corner from concave into convex in the interested angle, that is, "bending the corner". As the obstacles are made of hard paper, it is easy to make. So it would be able already to measure the distance to the limit of the obstacle.



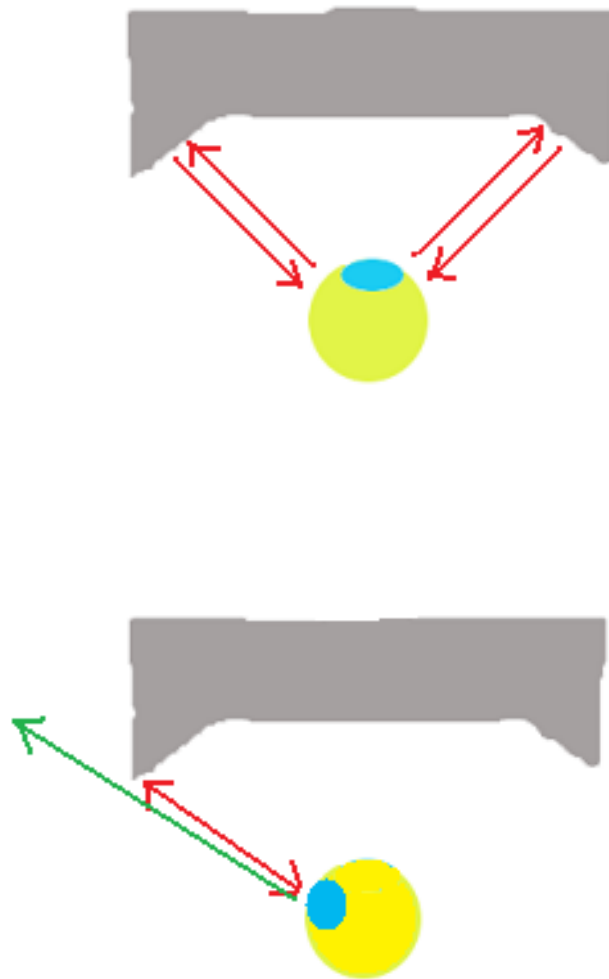


Figure 19: Enabling for bounds measurement

This resolution will only work while the robot is approaching the obstacle in a suitable distance to face the bound in the correct angle (that is, right angle), which will always happen as the robot is ordered to stop at the same distance (quite close) when approximating to the obstacle. Experimentally, all kind of situations occurred: Sometimes working and



sometimes not, due to some factors, such as the already mentioned considerable imprecision of the sensors and wheels, which induced the robot stopped in a wrong distance from the obstacle and not turning the exactly angle requested while sensing the obstacle perimeter.

Other aspect to change would be to choose another material for environment construction. The obstacles were built with white hard paper, so it searched for a type of hard paper more granulated, so that the grains in the surface had the function of deviate the sensor ray in order to go back. The more granulated surface, the more possibilities the rays go back to the receiver.

However, the granulated hard paper found it was not good enough as the grains were not big and numerous enough, so the trials did not work. After all, it is a hard paper. But the idea is worth to be considered in future applications.

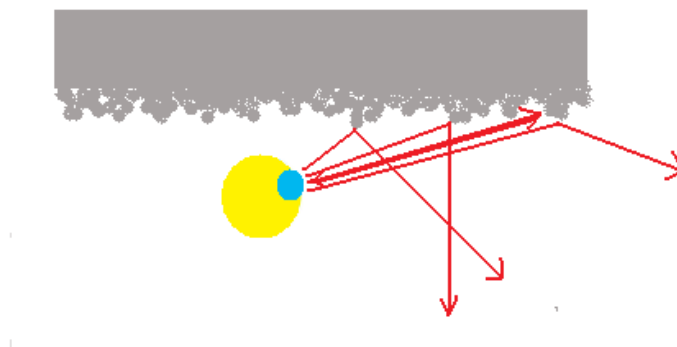


Figure 20: Obstacle's granulated surface





Figure 21: Granulated hard paper

The last resulting possibility is just the opposite, to use a polished material so is able for the maximum reflection possible (for example, in this case the aluminum, when polished, is a good option), or just a mirror. With the maximum rate of reflection it is increasing the possibility percentage to receive back the signal. It should combine this option with another one, as the burst infrared sensor. In this case the obstacle's bound should not be allocated too far from the robot sensing, as if the angle formed in the corner between the surface and the robot is extremely narrow, it will be impossible to measure. In the experiments, without any combination with other alternative, did not work.



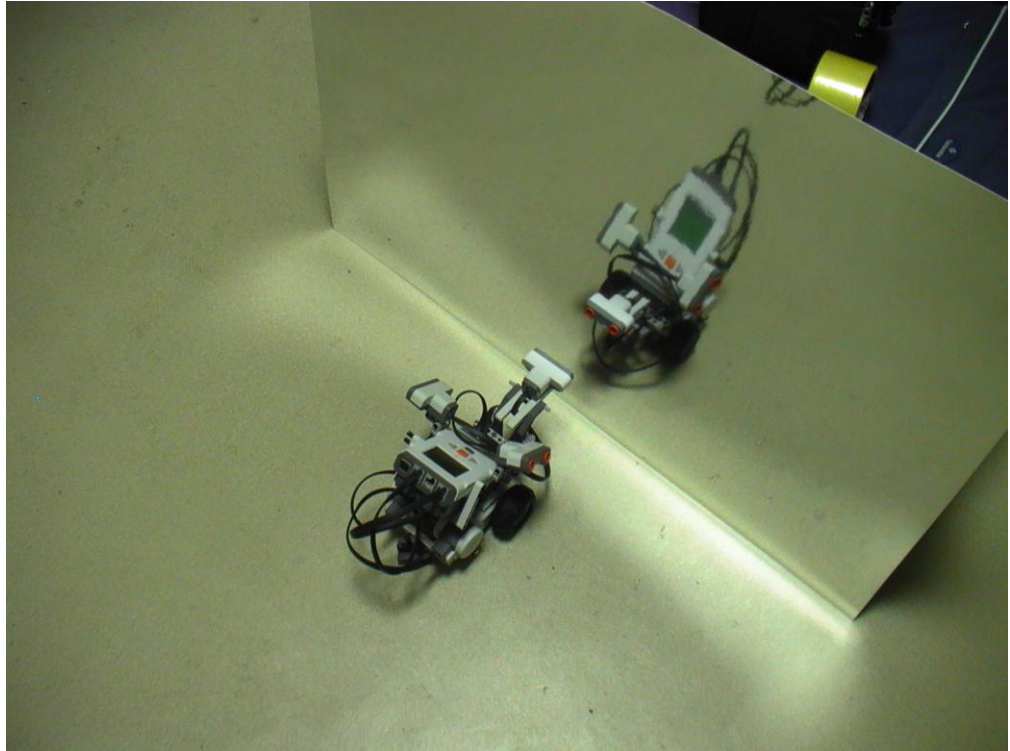


Figure 22: Robot dealing with mirror-type obstacle

Eventually, it can claim the first option is the most appropriate, so it takes into account implicitly, that is, assuming the “bending corner” in the next explanations.

Taking the successful trials, it proceeds to explain the calculations carried out in this technique:

Knowing the distance to the obstacle and registering the angle in which the bound is detected, it can determine the distance to the bound by simple trigonometry.



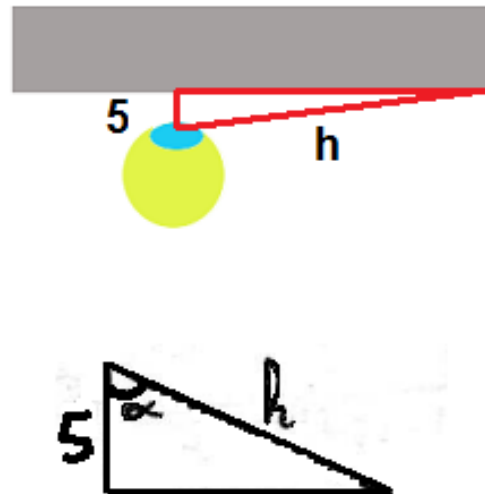


Figure 23: Distance-to-bound calculation.

$$h = \frac{5}{\cos(\alpha)}$$

Ec 1: Distance-to-corner equation

After determining the direct distance to obstacle's corner, it uses that data to get the distance the robot would have to travel parallel to the wall in order to avoid the obstacle.



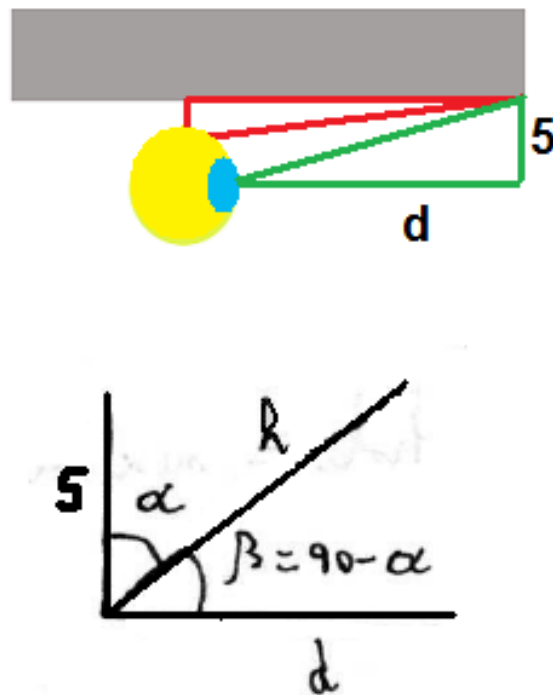


Figure 24: Calculating distance to travel

$$d = h * \cos(\beta)$$

Ec 2: Distance-to-travel equation

To result a proper distance to travel, it has to consider a couple of things: The difference of measuring, that means, in the figure 24 it can appreciate that when the robot is sensing the bound, that is not the same position that when the robot is parallel to the wall, preparing to avoid it. It is because the sensor is not located in the center of the robot but in the



front face. Anyway this is a small distance that could be omitted if involved in good conditions. The other aspect, more relevant, is that the total distance to travel is the calculated one plus the robot length, as minimum.

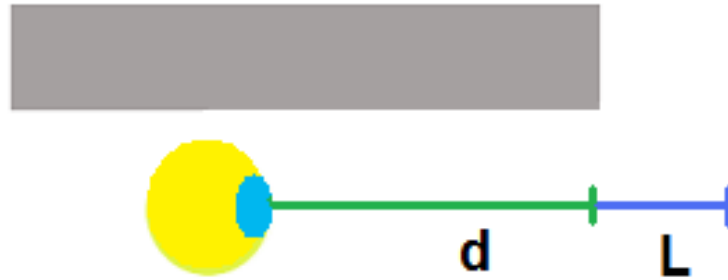


Figure 25: Total travel distance

Analogously, when the robot is facing the obstacle's side wall, it estimates the distance to travel, though this time could be more efficient (faster) to travel a certain length and check if the wall is finished or it must go on.



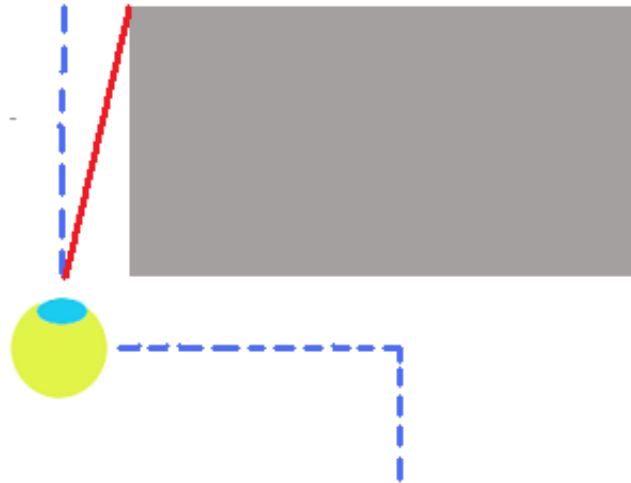


Figure 26: Travelling side wall

Before deciding a suitable distance to face the obstacle and a basic angle with which turn and check repeatedly, it did a little theoretical study to find out if the relation distance-angle is appropriated to be chosen for practice and calculus.

After trying with some different numbers, it was chosen 5cm distance and an angle of 10° .



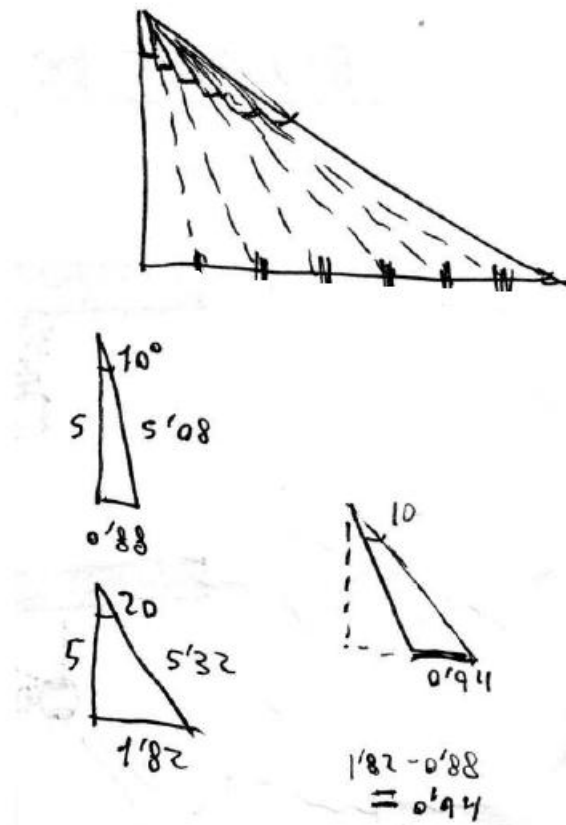


Figure 27: Calculating each shot separately

When the robot performs the calculus is directly did and just the last and useful data is overwritten and saved, but now, it is doing step by step, solving each triangle separately, with 10° angle and an original distance of 5 cm, which is the side of the first triangle but of course is different when facing the next ones, so it must base calculations in the previous triangle, by connecting the results.



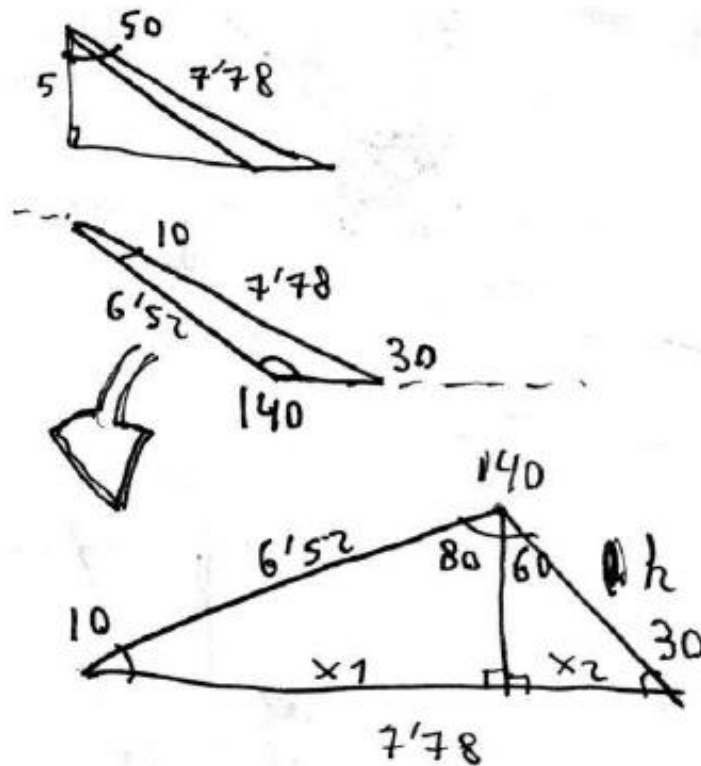


Figure 28: Triangles calculations

$$\cos(10) * 6,52 = x1 = 6,42$$

$$x2 = 7,78 - 6,42 = 1,36$$

$$h = \frac{1,36}{\sin(60)} = 1,57$$

$$5 * \alpha = h + 2$$

Ec 3: Relation distance-angle



The result of this short study is that with these data it confirms that there is a little enough difference in increasing the whole radial turn, the obtained relation is that if the angle is multiplied by five, the distance is only increased in two centimeters, so it is no problem. It takes this data for implementation in the robot application.

A last factor must be mentioned about this algorithm. Implementing this program, the robot is able to estimate (when calculating the obstacle's corner) if there is a possible way, in case of existing a "hole" between two close obstacles by calculating the hole width and if it is large enough for the robot size, this variable will be added to the decision of which way should be taken.



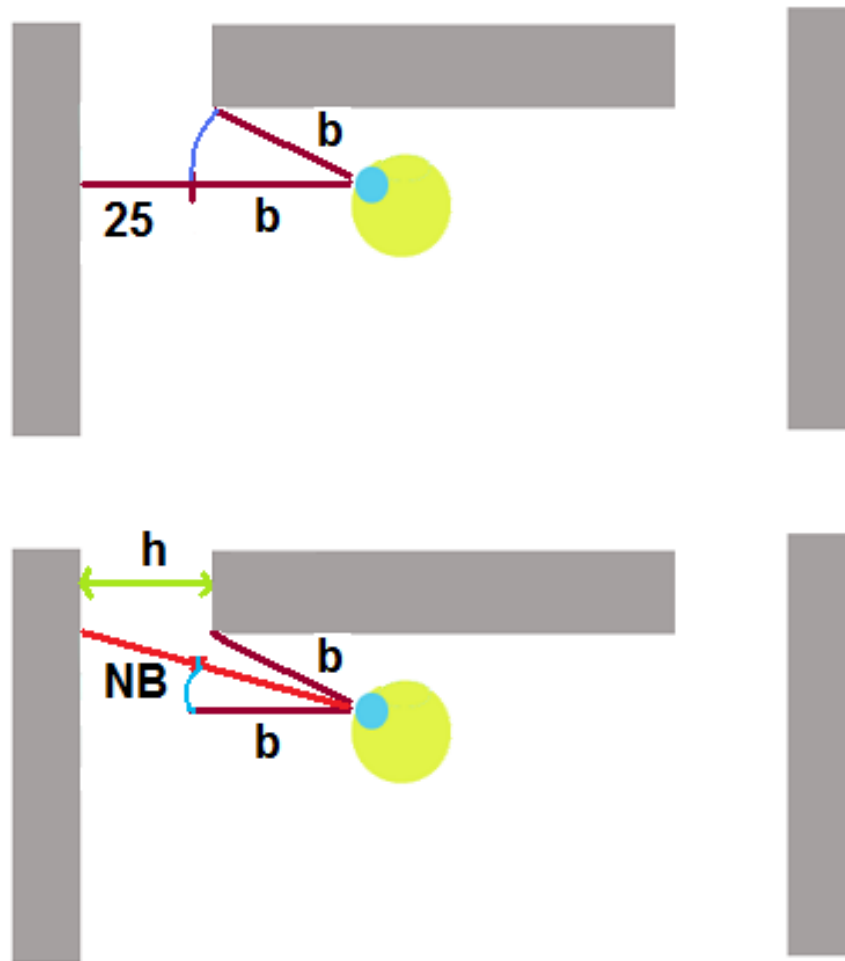


Figure 29: Calculating the hole

The robot registers the bound distance (b) and just the next one (NB), so it can already estimate approximately the hole's width (h). It must be at least the sum of the bound distance plus the robot length in order to be accepted and approachable.



$$NB = \frac{b + 25}{\cos(90 - \alpha)}$$

$$h = NB - b$$

Ec 4: Hole's width resulting

But as already mentioned before, this technique only can work in so specific conditions, which could not be made successfully. That is why it has to think of other option to put in practice.



Easy Avoidance

After some considerations it concluded that best way would be to omit these calculations and to simplify considerably the algorithm, so the robot performs just deciding one way and trying to surround the obstacle over there, if not possible, it change to the other direction.

Firstly it tries to surround by right side, and later by left. After turning, if there is no front obstacle, it moves forward and checks the wall each certain distance (40 cm predefined). This way may seem too simple, especially compared with the last one, and lacking of emotion, but unfortunately with the available resources there is no other way to obstacle avoidance in efficient way.



Figure 30: Robot facing an obstacle



The last algorithm tried to get the best way in terms of space, that is, travelled distance, forgetting the time spend in that task. In this more practical algorithm take the opposite approach by sacrificing the optimal way for avoiding the obstacle in the least time possible, that is, performing always, non stopping for calculations, only checking sometimes the wall in order to find the limit of the obstacle and the earliest time to turn and go on.



Figure 31: Lego Robot travelling straight towards an obstacle



The experiments carried out resulted successfully. Afterwards, it stepped forward by adding two more ultrasonic sensors to the robot. Easy Avoidance algorithm was modified to perform with three ultrasonic sensors: one frontal and two sides placed. In this conditions obviously the performing is better, faster; the robot surrounds the obstacles without turning to check the wall as the side sensor is already doing continuously.

Nevertheless the same problem persists: It unknowns the theoretical and exact distance to which the sensor is limited for detection, as based on the experiments it seems dependable of the obstacle's surface. Due to that the robot may find some tricky environments in which there are two side obstacles too (apart from the front obstacle, of course) and one of them is closer but it exists a "hole" to pass, and the other one is further but is connected with the front obstacle, so there is no space to surround. Then the robot was lead to wrong way, trying to avoid by further wall detected, and the other would be unconsidered. Finally the robot would not find the way so would think that there is no way possible, thus the robot would go backwards in order to try again from further distance.

This robot confusion would not take place in the previous algorithm, as the robot would be able to detect and estimate the "hole". But as it is not a feasible algorithm, it has to accept the simple one.



STRAIGHT FROM A TO B

The next algorithm is based on the previous one, taking it as background for accomplish the objective in this application:

The robot has to departure from a place that always is called A point. It has to direct towards the finishing place, so called B point. Of course there could be many different ways to get there, always avoiding obstacles, of course, but it decided to reuse the previous Easy Avoidance algorithm to face this challenge, so the robot goes always straightforward avoiding obstacles until arriving the final point.

The robot is given a coordinates about the point where it must get and then it calculates the suitable turn for the correct direction, after that, it moves straight.

In this case it can find the same problem that previously; for example, if the robot is travelling with 45^a deviation, the obstacles have to be right angled respect to the facing robot. If it is going with 30^a would have to happen the same, and so forth. Then, due to this problem, the environment is really restricted and whose orientation has to be preplanned before robot performance.

To get the B point, the robot, being independent the orientation, has to estimate also the straight distance to travel. Then the robot is registering the distance it is being travelled while on the way.



In this application may take place a tricky situation, which lead to the robot to a wrong final position. That occurs when the B point is enclosed in some connected obstacles, forming a structure of "U". Thus, when the robot finishes the counting and it thinks the total distance is already travelled and it is placed on the B point, actually is not.

The next figure can shed a idea about what it means.

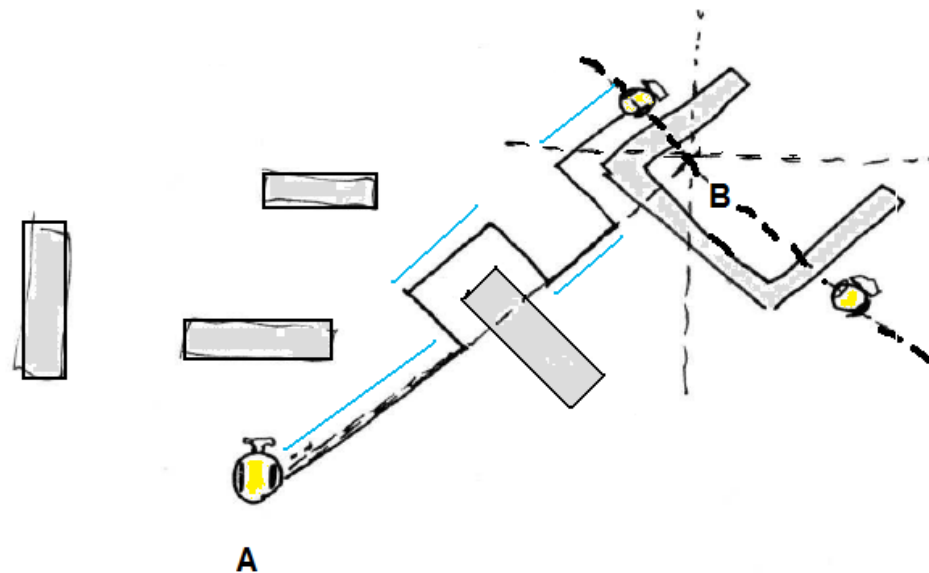


Figure 32: B point enclosed within tricky obstacle

To manage this challenge, it has added a new function in the algorithm, so called "WayCorrection", in which some breakpoints are saved in order to determine in which position the robot finish travelling



and get stopped.

Thus, knowing the position and depending on which one, it runs an action or other one, always focus on going inside the “room” where the B point is placed.

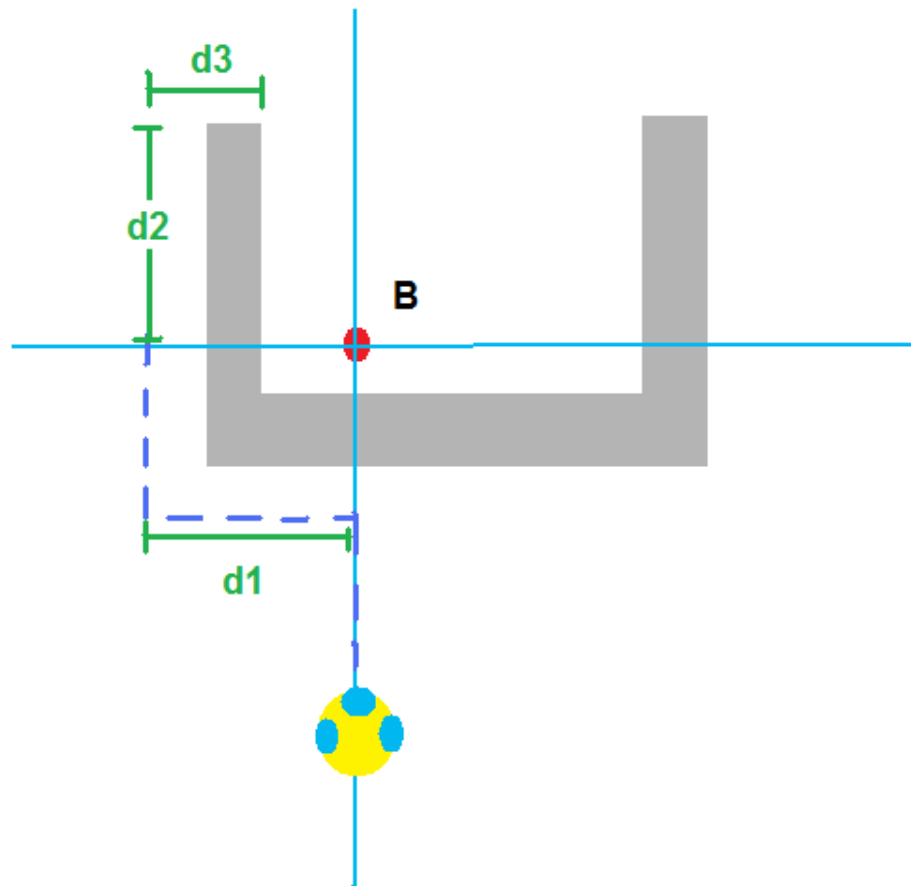


Figure 33: Estimation of the final obstacle

While the robot is surrounding the obstacle, it is measuring and registering the indicated distances in order to correct the way “inside” and get the desired point.



In case of a little bit more difficult final obstacle, whereas there is a hole to pass, the robot could get the B point.

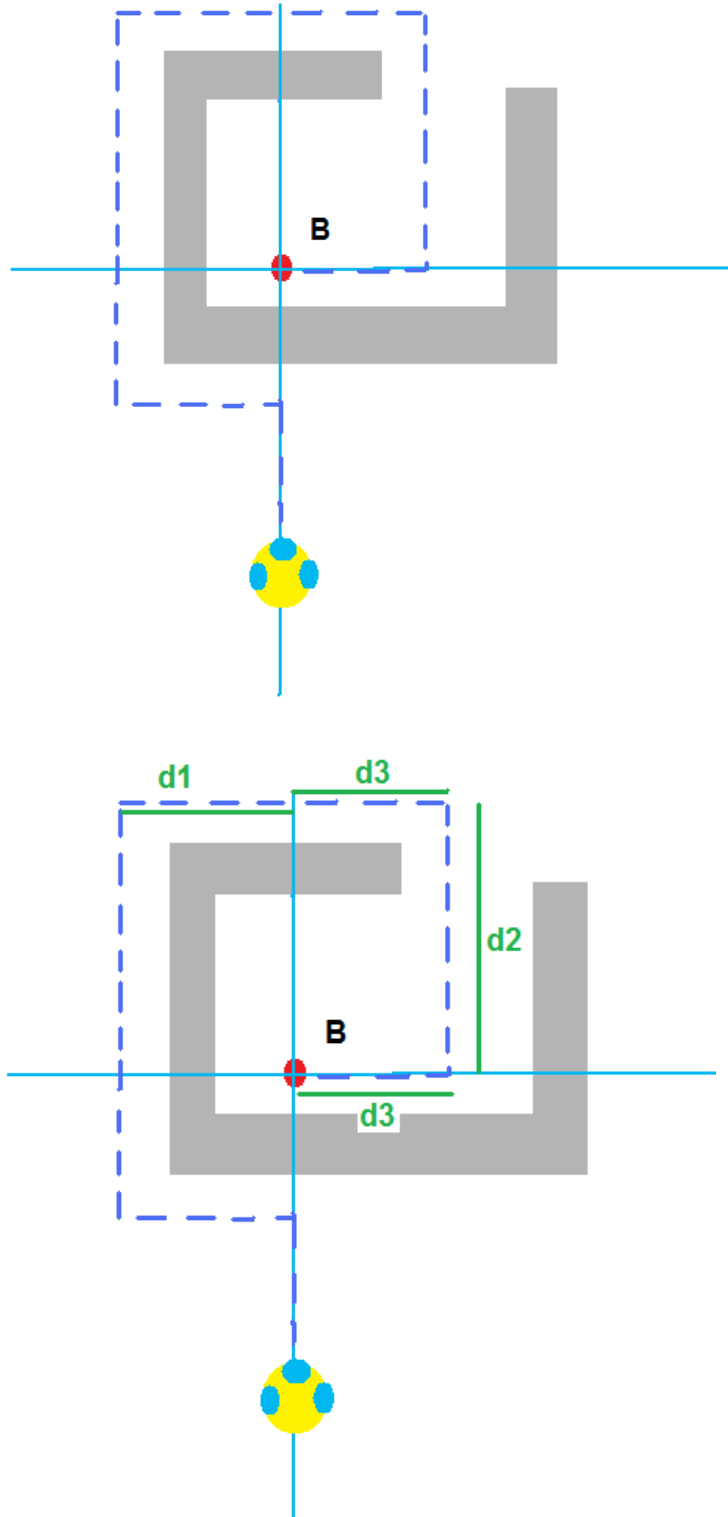


Figure 34: Travel distances to get B point



As represented in the figure, in this case (and the next ones), the robot has got three ultrasonic sensors to make easier the accomplishment of the goal.

In addition, for a correct and more careful performance, especially when dealing with the final obstacle, it has included a couple of extra functions, which have to keep the robot at the same specified distance to the wall when is surrounded along its perimeter. This implementation on one hand is avoiding that the robot might lose the following wall, and on the other hand it slows down the speed and efficacy of the robot behavior due to have to pay attention in keeping the margin respect to the obstacle.

The previous registered distances modeling the obstacle later are used to travel the “room” and to stop just in the B point location. In the experiments some problems were occasioned maybe because of the robot length and the deficient turns, but at its whole, the resolution is good enough.



FROM A TO B: THE LEAST X / THE LEAST Y

This algorithm is just a modification of the previous one. It keeps the background about obstacle avoidance travelling straight, but instead of going directly to the target, in some environments could be more appropriated to take another different strategy. Of course, as always mentioned, the obstacles must be well oriented.

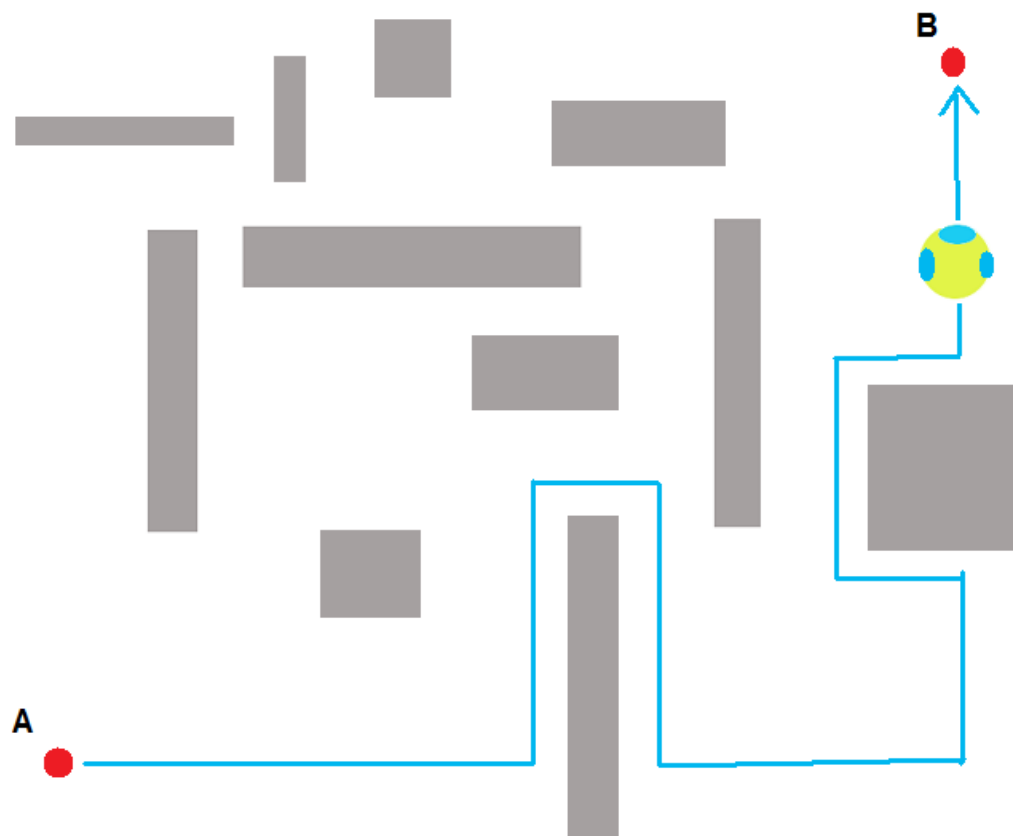


Figure 35: Example of AB-LeastX algorithm



Above it can see how the robot firstly travels towards the minimum X coordinate, that is, instead of going to the final coordinates X, Y at time, it divides the process in two steps, first try to accomplish one coordinate and later the other one. When the robot is located in the Y coordinate axle, then it starts to get the Y coordinate, which lead it to the destination

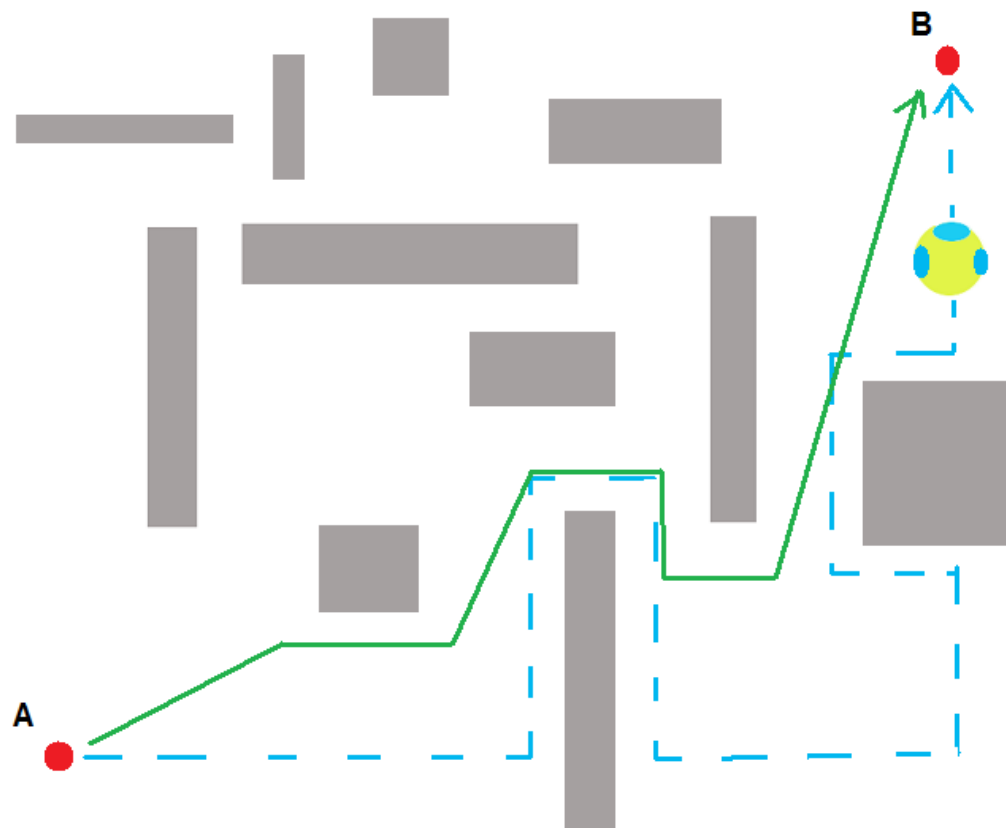


Figure 36: Optimal way

The approach is not leading to the optimal path, but in this case is quite similar, almost the best one, as it can see in the figure, the green line could be the optimal way as the other ways are actually closed because the



width robot is big enough for not passing through the holes and spaces among obstacles.

Of course, it is obvious in this type of environment it check that the previous algorithm, AB-straight, cannot be applied because the obstacles involved in the robot trajectory are not correct oriented.

In the next figure it show the analogue travelled path with AB-LeastY algorithm, which treats about approaching the Y coordinate before getting the X coordinate and consequently the destination.

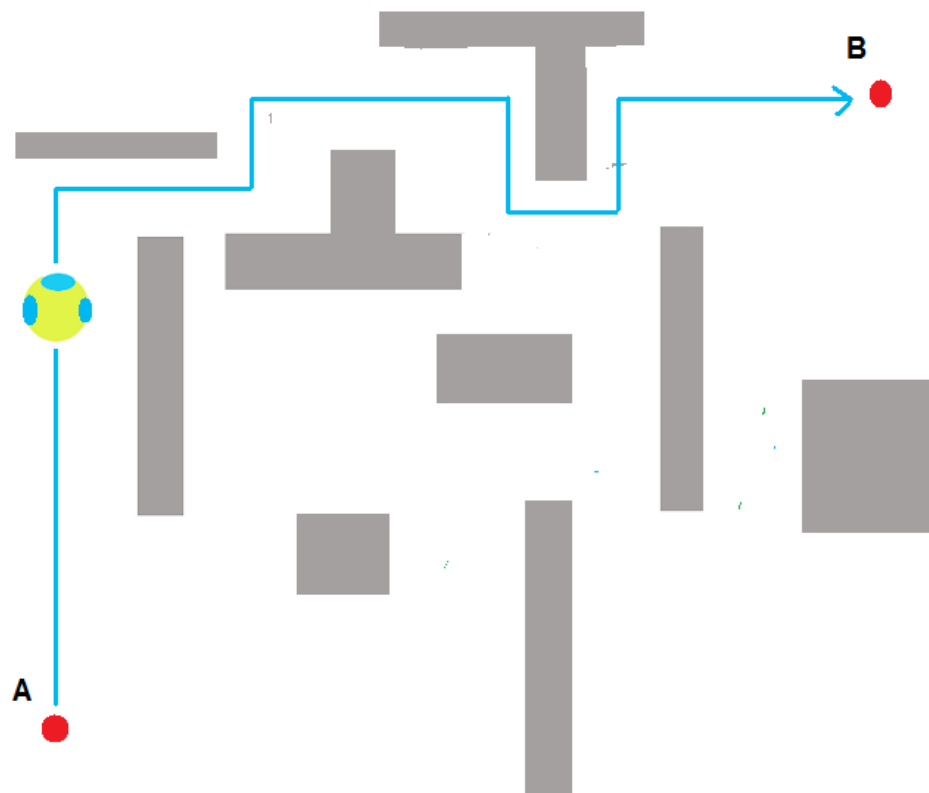


Figure 37: Example of AB-LeastY



These are just different manners to manage this kind of objectives and environments. Arbitrarily, the robot will always try the right path and later the left one, but it can be changed in the source code, as sometimes can be more or less convenient to give priority in one or other shift.

It does not forget that here also the tricky final obstacle can take place, so the corresponding functions are implemented.

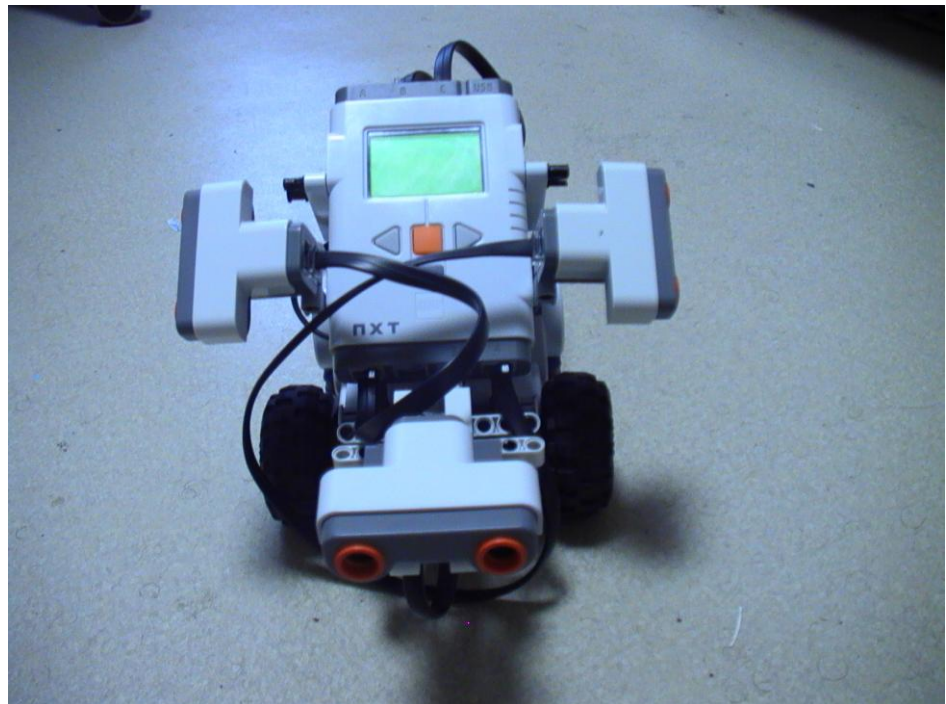


Figure 38: Lego Robot with three ultrasonic sensors connected



HUNTER

This algorithm well could not belong to this project but it considered also interesting and a typical application in general mobile robotics, so it decided to try programming somehow anyway, despite knowing is not possible to put in practice as it is necessary other robot (the prey), which is not available.

It is a simple game or competition, but complicated program, especially in this case, as the robot does not know the environment in advance. Usually this event is planned by making a simulation and the robot does know the environment layout by having a file in its memory or by mapping. Thus, the performance is easier and the robot can be focus more strongly in the hunting strategy (or prey escape).

However, as the robot has to deal continuously with new and unexpected obstacles, it has to take the useful obstacle avoidance algorithm and later it must implement any strategy for hunting which runs according to the hunting process, that is, they have to have a good level of compatibility each other in order to obtain a efficient robot behavior in managing the environment while pursuing its target.

Regarding the rules, as simple as above mentioned, the environment is full of obstacles and the hunter has to find the



prey and pursue it until catching (that will be considered as a result over 85 in the signal reception of the light sensor), then the hunter wins. But if the hunter does not find or get lost his target, as surely the prey would get the initial point (hunter exit).

Thus, the process and features are as follow:

Firstly, talking about the strategy and behavior it chose a motion prowling-like. That is, in this case, shown through a kind of zigzag performance. It had to look for and decide how the hunter would search for the prey without knowing the environment and without leaving in the open the exit point where the prey can be saved and become winner.

Certainly the hunter strategy could well be to stay very close to the exit point and waiting for the prey approaching, but it would not be right and fair, so it is not allowed; the hunter has to hunts.



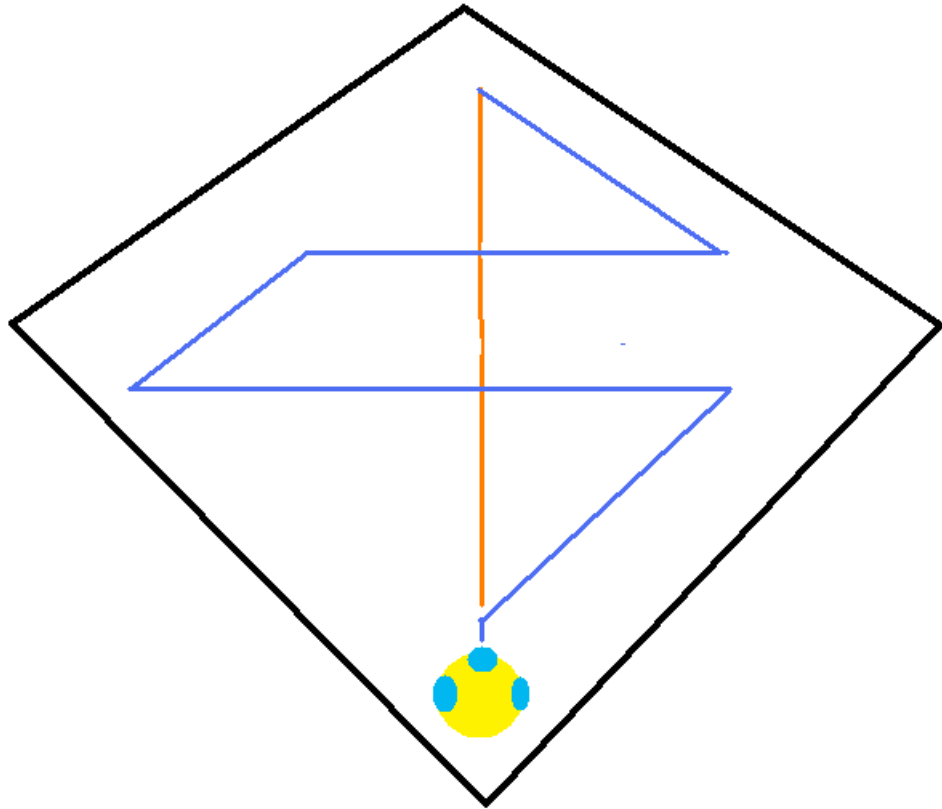


Figure 39: Zigzag Hunting

Here is shown the scheme of the prowling trajectory to look for the prey. It decided use this kind of movement because thought all environment is not covered, but the most. Thus, the hunter has many possibilities to see the prey in any time.

It is assuming the hunter always starts in a environment's corner and already oriented, that is why the perspective of the figure above..



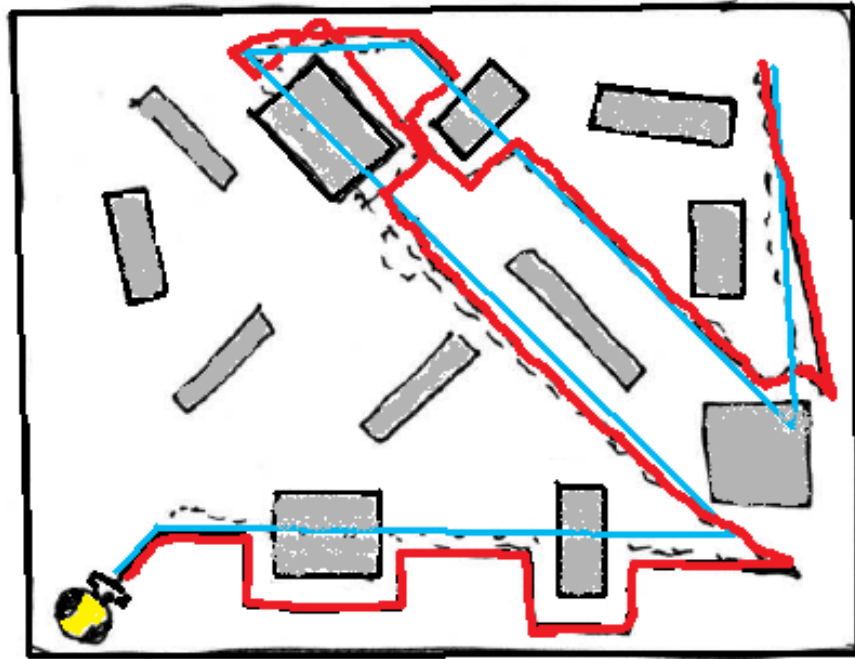


Figure 40: Zigzag hunting in obstacle avoidance

Above it can see a example of hunting with zigzag prowling. The blue line marks the original way the robot has to follow and the red line represents the real path travelled by the robot, as it has to go on avoiding obstacles. Here some incongruities take place. For example in the obstacle located most at the left side, the robot is surrounding it by the right wall and when finished it has to turn and follow the path by going back by the same way. That looks like no sense but it is just a discovered detail about some incompatibility between both behaviors. Other matter is when the robot is surrounding a obstacle and before finishing it is the time to turn to



go on the zigzag, then some deviation occurs and the next orientation is not already exactly the requested one.

For the robot does a zigzag movement suiting the environment, it must know the size of the square, so it would be the only detail about it that the robot would know in advance.

The program does not allow so much complexity, so there are some disadvantages. Unfortunately the sensing is discrete, that means the robot cannot find the prey in any moment, so there are checkpoints where the hunter can see the prey and the other times do not. For example, whereas avoiding obstacle the hunter cannot pursue the prey though saw him.

The most important checkpoints, where the hunter takes advantage to look around, are shown in the next figure.



There is an important matter not considered so far. How can the robots recognize each other without confusing with the obstacles? Well, as the robot's surface is very clear, quite white, a good option is to use black obstacles and a light sensor to differentiate the obstacles and the other robot.

In addition, if there were available two light sensors, it could improve the source code to able the hunter to detect what way the prey has taken in his escape, so the reaction of the hunter would be much better and it could do a more efficient hunting.

Then, in this application the robot has got three ultrasonic sensors and one light sensor, so it is using all connectors. However it unknowns the range of the light sensor in official data, as such information is not provided. Anyway, in case of short range, the most possible, there would not a great problem because both robots would have the same problem: Hunter needs to be close to detect the prey, and the last one the same for avoiding and escaping from the hunter. Moreover, the environment is not so large, so there has to be enough percentage for both to get victory.

In the next figures it shows the Lego Robot with the mentioned black obstacle to make sense what has been explained. But in the figures the robot is just posing, without the light and side ultrasonic sensors.



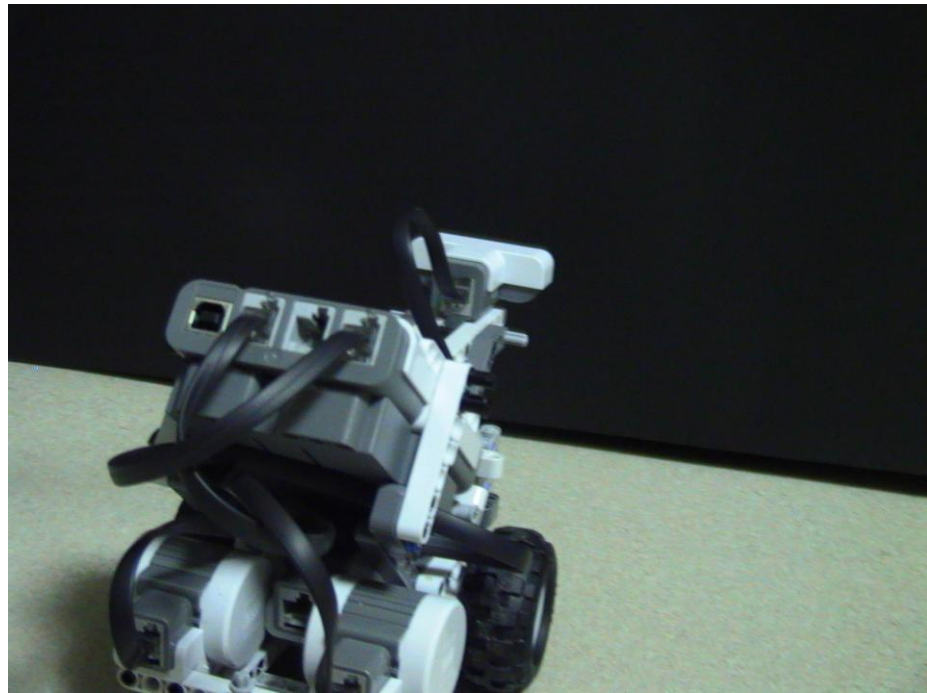


Figure 42: Lego Robot facing a black wall in hunting behavior

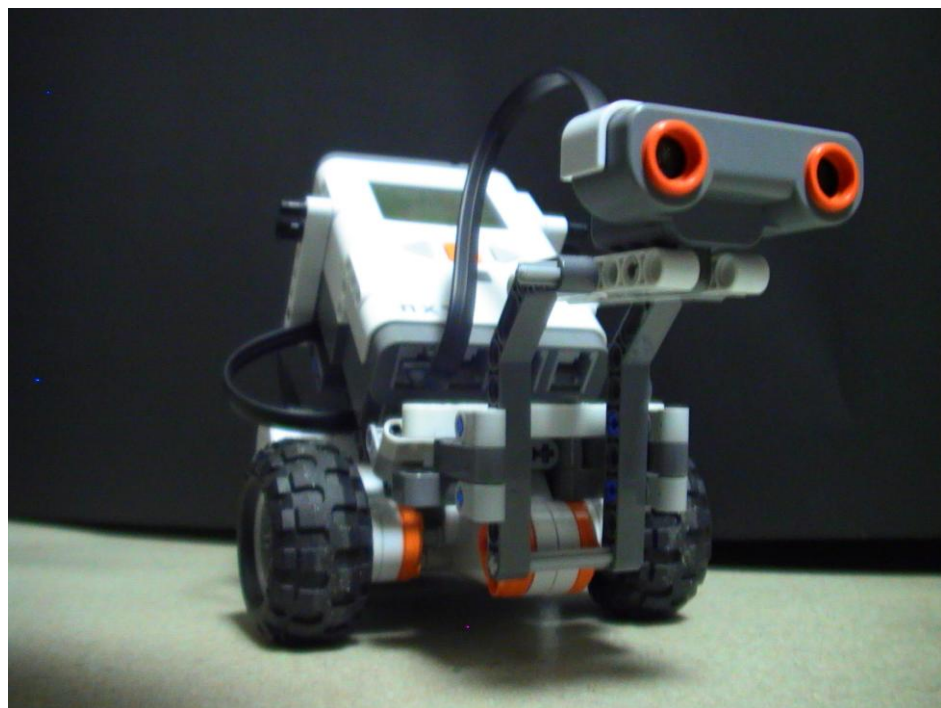


Figure 43: Lego Robot in front of the black wall



Line Follower

As mentioned, this application has been developed mainly because it is a unavoidable example in robotics projects, not because of his level of difficulty; and anyway is another kind of environment where the robot has to manage by trying to accomplish an objective, so it considers suitable for this project.

To build the environment it has used, as usual, white and black hard paper. The circuit is not quite complicated, just following the regular ones, but adding some close curves with right angles in order to check if the robot can follow it with a good performance.



Figure 44: LineFollower circuit





Figure 45: Right-angled curve

It could program this application in many different ways depending on the approach, available sources or the sophistication it desires to get. For example, it had been much better to have disposition of two or more light sensors. Thus, the robot could follow the line faster without reducing the efficacy.

But it disposes of one light sensor and according to that it has developed two different programs, in practice quite similar anyway.

The first one is the simplest possible, taking as limit the mean value of the light sensor, it acts moving the motors independently, so the robot can follow the black line without complications if it is well defined and there is no tricky places.



The second one is a little more sophisticated, using exceptions and the LCD, but despite of differences in the code, in essence it is the same.

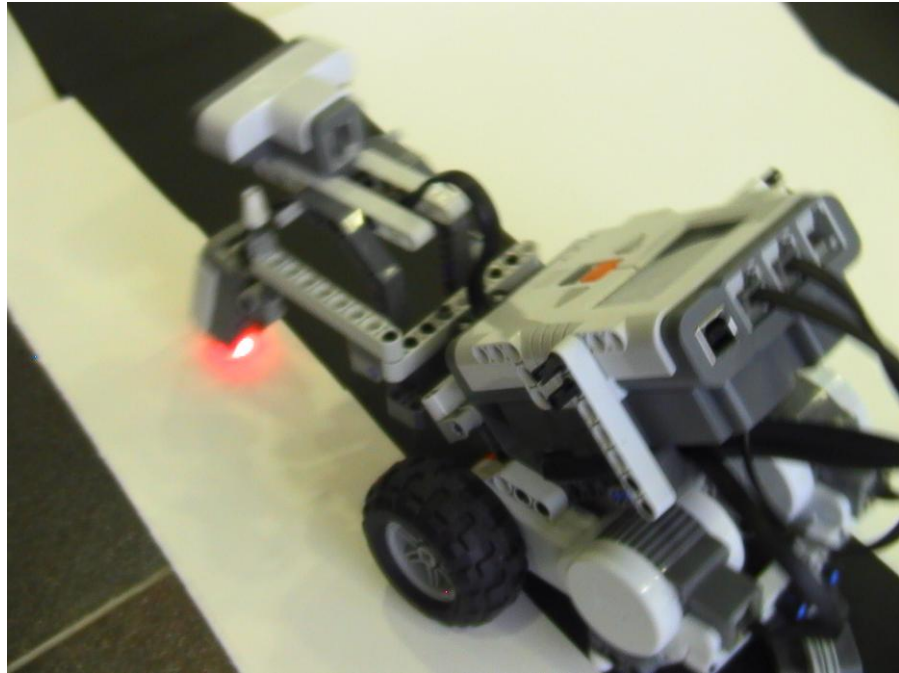


Figure 46: Robot following the black line

The performance in the experiments was very successful. Even the robot got to move quite faster than expected.

In the next figure it can observe the good performance of the light sensor, the bright coming from the white area respect to the black one is easy to appreciate.



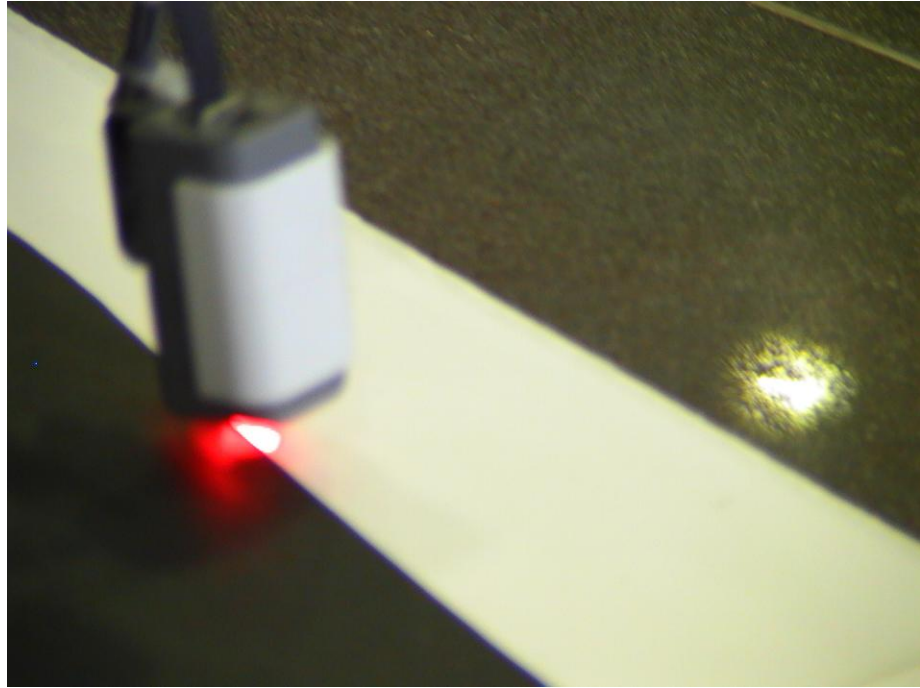


Figure 47: Light sensor in action

However, if the light sensor is not so close to the floor, the action results quite more deficient, so the range of actuation is really short, at least in this case in which it needs a totally clear difference between both regions as it is working in the limit of the line.



MAZE: TREMAUX SOLVER

The maze solver robot is other good and interesting point of this project. It is being also a well known application in the robotics fans world. This application takes places in many competitions with standard or non-regular rules and conditions. This specific project is not under restrictions of a robot competition (though if desired, it can follow some rules or standard specifications if being helpful or in order to have a previous guidance), so there is a wider freedom to decide and act.

For this kind of challenge, the robot needs some strategy to find the way out of the labyrinth. The most usual approach is the robot follows all time the right wall or the left wall. This is too typical, so it has been omitted. Instead, it has preferred to use a different algorithm more useful, as following the wall works in many general mazes, but there are some labyrinth that are prepared for that this method fails, for example when the exit is not located in the external wall but in one of the internal ones.

For these cases there are other not so common algorithms, such as the so called Tremaux's algorithm, which is used here. This method requires memory, because the robot has to memorize the information about the maze, for instance the way already travelled or the way not visited yet. For that, the maze is divided in cells.



To put in practice this application it has built a labyrinth (after design), and as in the previous other environments, it has used the white hard paper.

The chosen design is a 10x10 Matrix, which is divided in 25x25 cm cells, such measurements are the robot length.

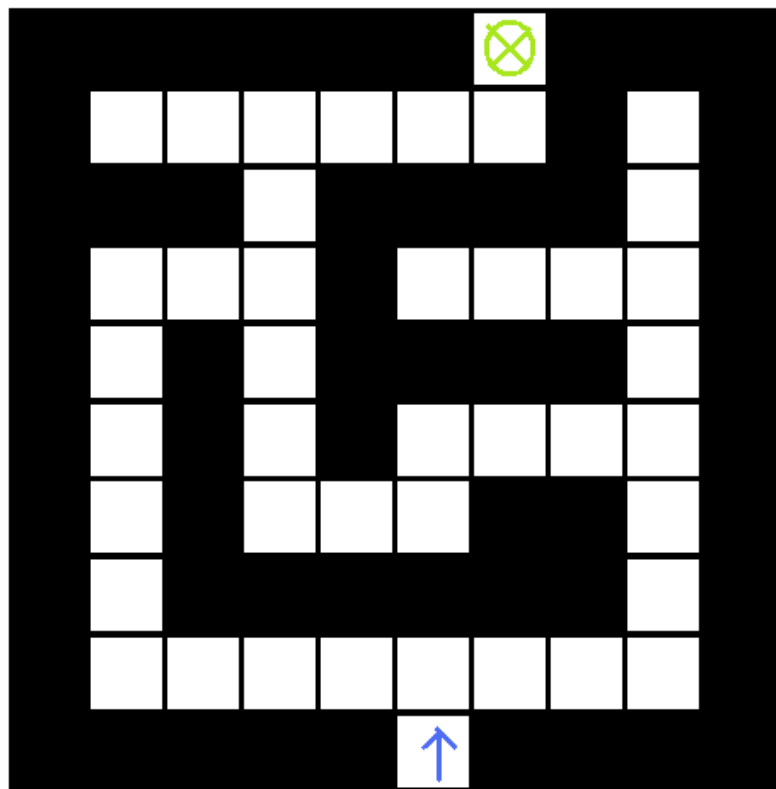


Figure 48: Maze scheme divided in cells





Figure 49: Real Maze

Above it can see the result of the maze construction. Cells are 25x25 cm, as the design requires. However in the experiments appeared a problem, the measurements are a little tight, so sometimes when the robot has to turn or it is just a little deviated, it gets crashed the walls and then cannot turn all ordered degrees or it just keeps blocked. Apart from that, the performance is quite good, showing a efficient behavior representative of the program algorithm.

All curves are supposed right angled, so every turn should be perfectly corresponding to, but the friction with the hard paper and the tight corridor cause bad turns, so the robot many times must be corrected manually in order to go on the way.

Daniel Egido Sánchez de Vega



The maze solver robot always advances step by step, that is, cell by cell, so it travels 25 cm and checks the three directions: right, front, left; with the three ultrasonic sensors, in order to decide the next step.

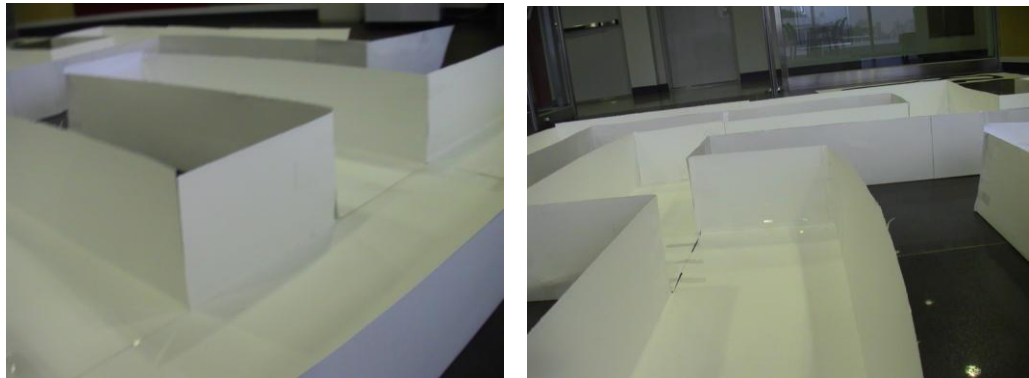


Figure 50: some maze paths

Next, it is going to explain about the Tremaux's algorithm and its technique for programming and to understand how it works.

First, it must define four "directions": north, west, south, and east. These directions do not have to coincide with the actual four winds. Let's call the direction of our robot at the entrance "North," and the other directions follow the usual convention. The current "direction" will be held in a variable called *MyDir*. For each direction must be assigned a number to define it: North = 0, West = 1, South = 2, East = 3,

It needs also to know which direction is to the left or right of 'Dir'

```
static void Left_of (int dir)
{
    LeftOf = (dir+1) % 4;
```

Daniel Egido Sánchez de Vega



```

    }

    static void Right_of (int dir)
    {
        if (dir == North)
        {
            RightOf = East;
        } else {
            RightOf -= 1;
        }
    }
}

```

Moving one maze unit forward, that is, one cell, corresponds to moving one "position", that means, changing the robot position in the map by 1. The map is stored in a variable called *Map* which holds a value between 0 and 3 for each maze "position":

Value = 0. This corresponds to a map position which it did not visit, or those which it cannot reach at all.

Value = 1. This corresponds to a position which we've passed already; in addition, it has already tried all possible routes emerging.

Value = 2. This corresponds to a position which it is already visited, but from which it have not tested yet one emerging route. For example, coming at a "T"-intersection, it needs to choose one path. The other path is still unvisited, so it must mark this position in the map with a value of 2.

Value = 3. This corresponds to a position in which two emerging routes have been left unexplored. This occurs when it comes firstly to a four-way junction. It will take the right path, leaving the other two for the future.



We really need to use this method, to remember the value of the cells, as for instance, when the robot finds itself in a dead end, it needs to backtrack through its own path until the robot reaches a position in which an unexplored path was left.

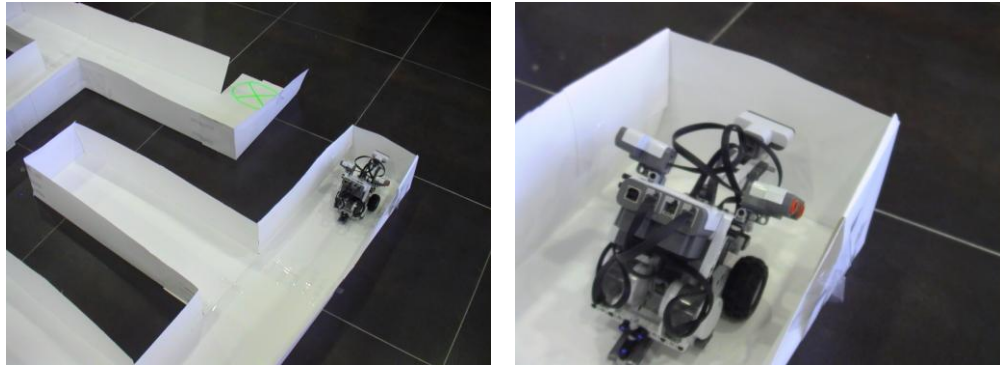


Figure 51: Maze solver dealing with a dead path

For an easier backtracking procedure, it marks these positions on the map with a value of 2 or 3, making it faster to go back to these. The current position in *Map* is stored in two variables, *MyPosX* and *MyPosY*,

```
int Map [SIZE_X] [SIZE_Y] ;

int MyPosX = 0,

MyPosY = 0;

int MyPath[SIZE_X*SIZE_Y] ;

int PathCounter = 0;
```

It defines two more useful macros that return a relative map position at a particular "direction":



```

static void RelX(int dir)
{
    if(dir == West)
    {
        RelPosX = -1;
    }else if(dir == East){
        RelPosX = 1;
    }else{
        RelPosX = 0;
    }
}

static void RelY(int dir)
{
    if(dir == North)
    {
        RelPosY = 1;
    }else if(dir == South){
        RelPosY = -1;
    }else{
        RelPosY = 0;
    }
}

```

Now it can find out what is around robot in the maze map. The current position is: $Map[MyPosX][MyPosY]$,

In front of the robot is: $Map[MyPosX+RelPosX(MyDir)][MyPosY+RelPosY(MyDir)]$.

And to the right is: $Map[MyPosX+RelPosX(RightOf(MyDir))][MyPosY+RelPosY(RightOf(MyDir))]$.

And to the left is: $Map(MyPosX + RelPosX(LeftOf(MyDir)))[MyPosY + RelPosY(LeftOf(MyDir))]$

It does not want to refer to positions outside the array bounds. It defines the maze array to be two maze units bigger in each dimension



than the real maze. Then, at the start of the program, it sets the outer "frame" of the array to a value of 1, and the rest to 0:

As it considers a value of 1 to be an "exhausted" position, it will never try to reach it as the robot moves around the maze. This will also prevent it from trying to move out of the entrance if the robot encounters it again, for example if a loop exists.



Figure 52: Maze solver in a crossroads

The first lines of the code check the distance to the wall at each of the four winds. If the distance is smaller than some predefined value, then it decides that there is a nearby wall in this direction. It assumes that all accessible directions lead to an unexplored position and later it checks the map at each one of these directions. If we indeed find that cell



unexplored, it continue deciding to go there next. Otherwise, it must decrease the number of "waiting" paths from the robot present position, as well as from the position we look at. Really, the only way it is going to encounter a visited position while traveling in an unexplored route is if it is closing a loop, so the previously traveled position must have had more than one path available when the robot was there. And of course, it has to take into account that anyway it must decide a priority in choosing ways, according to that It has to mention this source code is a "right-hand rule" code" If all paths are accessible and unexplored, the program preference is to first turn right; if not, the next option is to move forward, and the last option is to turn left.

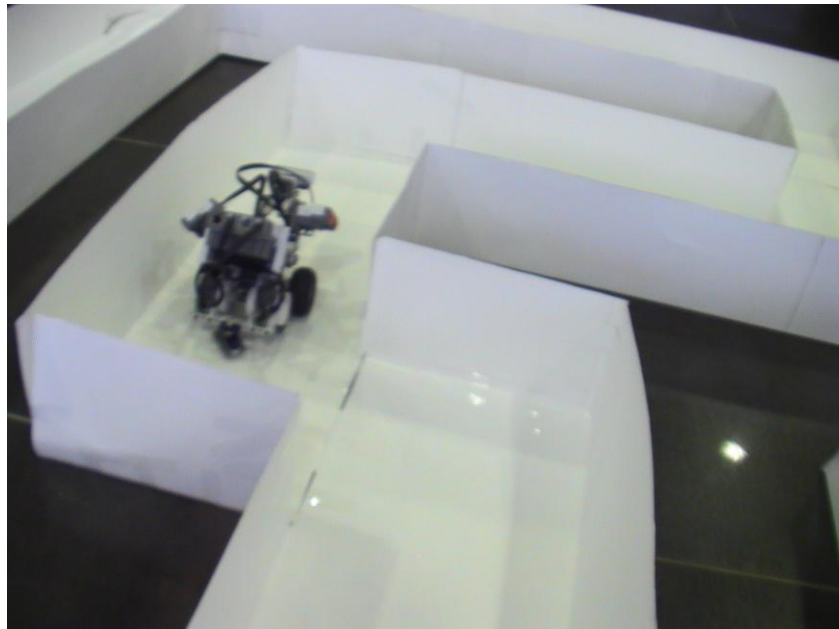


Figure 53: Maze solver traveling the labyrinth



When it has decided finally about the next step, the robot rotates to that direction and move forward:

Once it has reached a new emerging route position, it returns to the start of the “while” loop and it looks and checks again for a the next step, a path to take.

In the maze sample designing and tried, looking the robot performance it could say that it is moving by following right wall, but it is not true, due to previously mentioned this algorithm is applying the “right hand rule!, but anyway it could be changed and it would being the Tremaux’s method.

A better and clearer behavior would be seen in a maze with inner way out, as already mentioned, where following wall does not work.

Nest a couple of maze samples are shown for understanding.



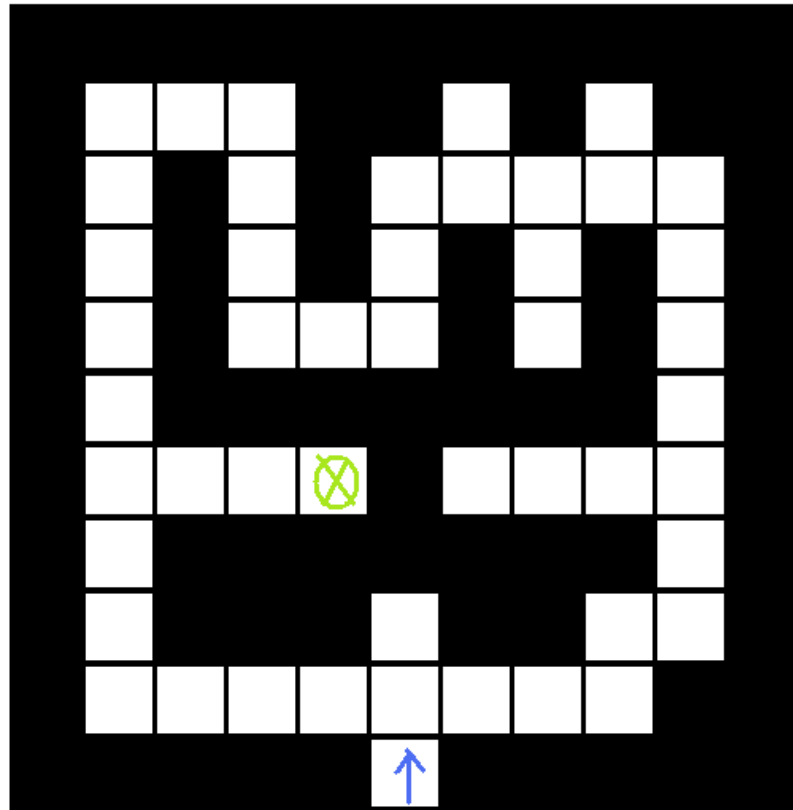


Figure 54: Maze with inner way out

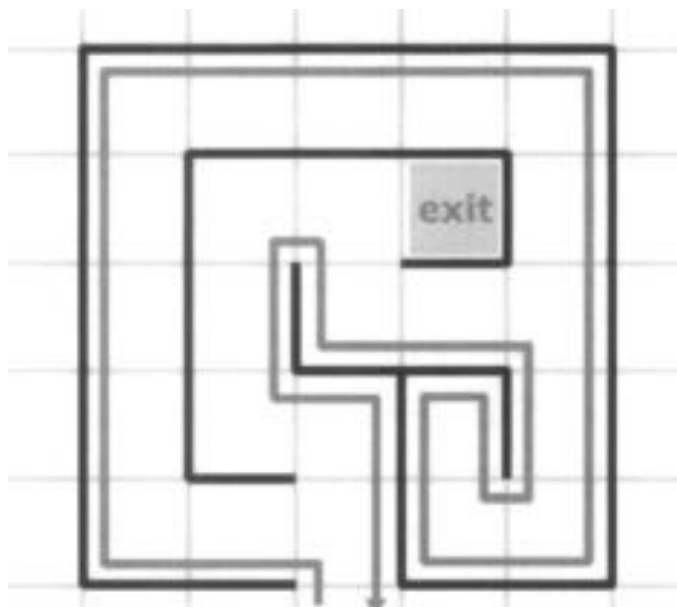


Figure 55: Maze with loops. The exit is inside.

Ref [DAMG07]



INTERFACE FOR ROBOT PATROL SYSTEM

As explained at the beginning, this subsection is an extra tool for the original objective of this project. It is about controlling or just ordering the robot to do something, via Bluetooth communication from a computer with a developed interface.

This application has been developed starting from a planned pattern due to the complexity underlying this program, which requires a higher level of JAVA knowledge.

The program is divided in two main sections: One is which will be downloaded in the robot (RPS_NXT), it contains the behavior, the navigation, that is, the method with which the robot know its own position and orientation, and the appropriated application Bluetooth-enabled for the reception of the instructions. The other one is which will be hold by the computer (RPX_GUI), enabling and activating the Bluetooth communication with the robot and supporting it, and the interface that the user can manage and with which can control the robot.

Due to having programmed this application starting from a pattern, it can observe clearly the resulting source code is more structured and schematized, it is divided in more sections and subsections, taking a clser appearance to JAVA programming formalized style.

Daniel Egido Sánchez de Vega



But not only in appearance but also in the code quality, as it is required a more advanced understanding of JAVA, it having learnt new programming concepts such as EVENT or LISTENER, which make possible to achieve this application to be developed.

The RPS_NXT is divided in some groups containing the sensing system, the actuators, the server, some utilities and useful datas.

The RPS_GUI is composed of some folders with the interface system, some protocol and listener.

After downloading the first one to the Lego robot, the second one must be run as a JAVA application, for that, it selects the main file (called "UserInterfaceSubSystem") and it executes. It is shown in the next figure:



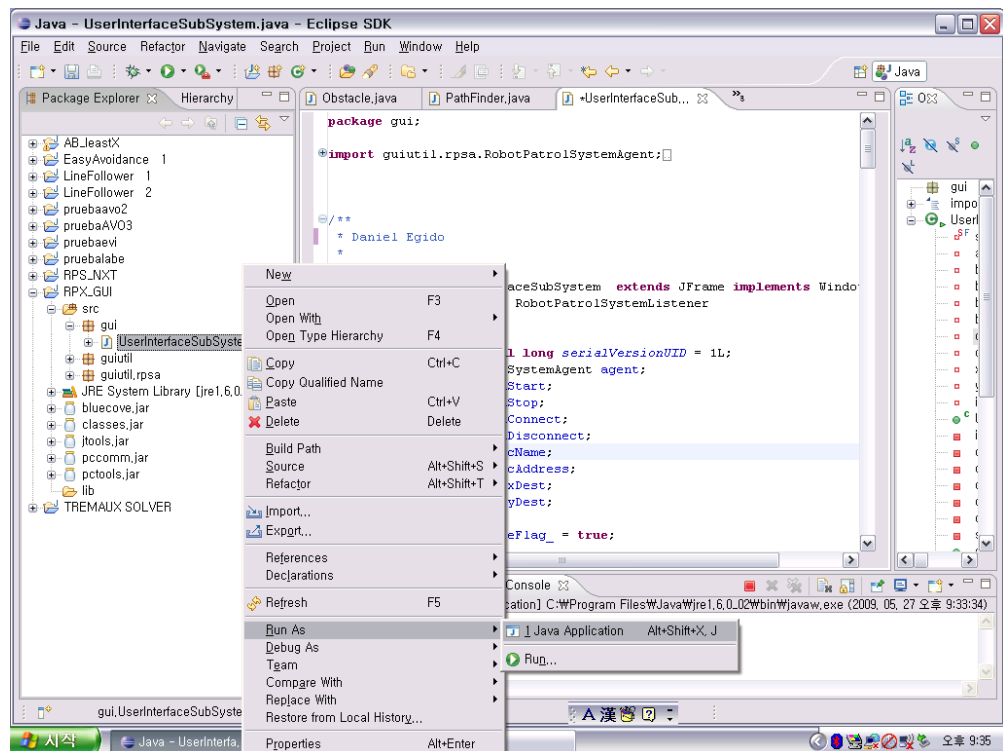


Figure 56: Running interface application

When done, it appears in the screen a small window, that is the interface, and it contains some information:

The name, the X and Y coordinates, where it must introduce the specific number in order to get the robot in some different position, it will calculate later the direction that it must taken to get there.

It is also included a kind of serial number, actually that is the address of the robot for the Bluetooth connection (embedded in the NXT). This data is found in the information about the NXT, in the brick.

Finally there are four buttons: Connect, Disconnect, Patrol and Stop, for the different options. Firstly it press the connection button



and continuing by selecting patrol, then the robot starts with the programmed and corresponding application.

In this case the behavior is not so much relevant, as what it wish to check is the good performance of the interface, so in the example practiced, the robot only go to the ordered coordinates and come back, and it repeats this motion until finding an obstacle in its trayectoria, then it just stops.

The experiments obtained good results, after some problems in the Bluetooth connection, which can became an annoying difficulty.

Next, some interface example with different coordinates are shown:

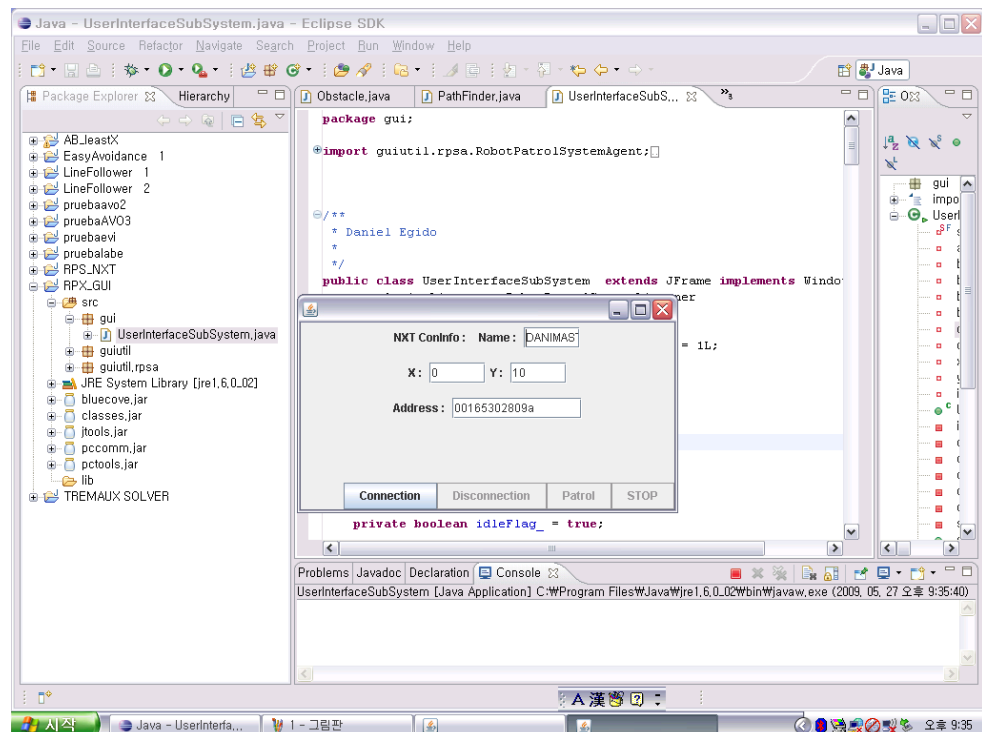


Figure 57: Interface within Eclipse workspace



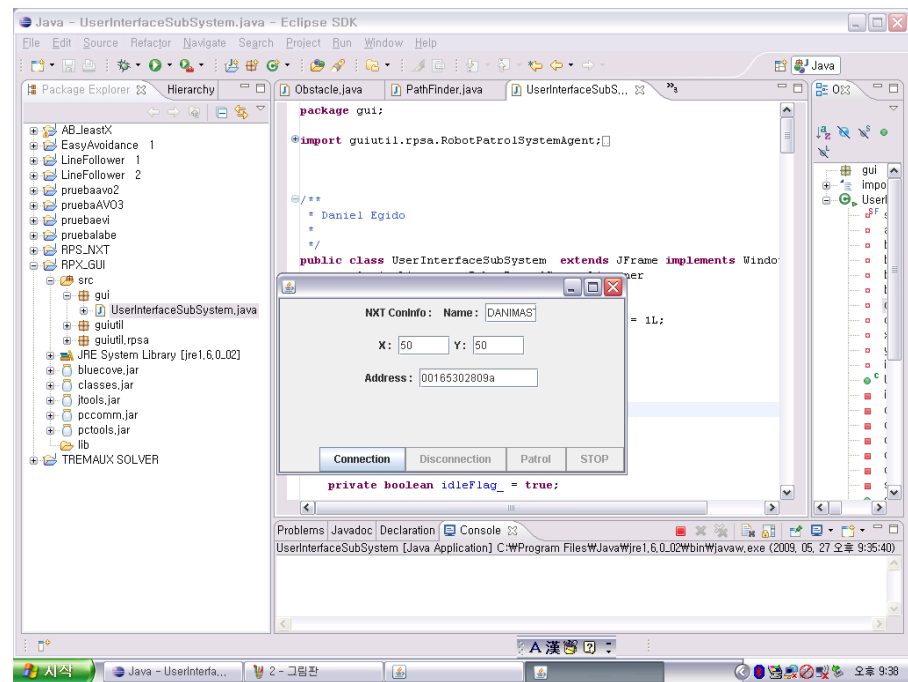


Figure 58: Interface with different coordinates

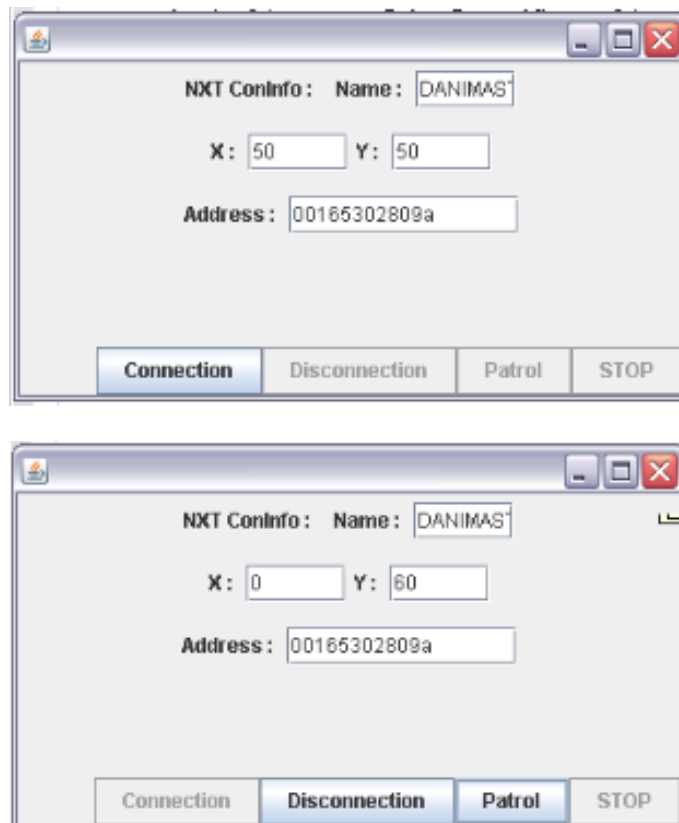


Figure 59: Interfaces with different coordinates



CHAPTER 4 CONCLUSIONS

As for the final judgment about this project, it must underline that it has resulted to be a worthy and interesting challenge, in which it has been able to deal with real problems in the robotics works, such as navigation and positioning, and to deep a little into this amazing field.

Despite the emerged troubles and resources limitations, it has developed a varied whole of navigation applications where it shows the capacity of the patrol robot to manage different environments in which it must accomplish diverse objectives and goals through common and alternative ways.

In the first stages it could not follow the planned schedule as should, but slowly it was reaching a better rhythm of working. Moreover, the lately resources availability and the slow process of JAVA learning caused an unbalanced performance during some time. But once set the basis and having understood the main concepts, not only about programming but also the different approaching for the algorithms and robot behavior, then it carried out a longer more efficient process until the last stage.



Due to that the trials could not be carried out before the last phase, the efficacy of the algorithms was not checked until then, consequently some methods were demonstrated not to have a good enough performance, so it had to correct and modify, even in some case to plan again the algorithm.

Anyway, though some of the applications seemed simple, other ones were developed with much sophistication, by providing new possible and alternative ways to get the goals.

After all, the put in practice obtained considerable results, with some interesting conclusions and consequences, and a adequate level of efficacy. So it consider in general terms to have achieved the objectives requested, even more satisfactory if it takes into account that it started from the very beginning: without previous knowledge about Lego robotics technology, Eclipse workspace tool, JAVA object oriented programming and so on.

That is why it appreciates and assesses at great level all learnt during the development of this project.



CHAPTER 5 FUTURE DEVELOPMENTS

This project is focus on a quite opened subject, as it is robot navigation, so depending on as it is conceived it might add and include many kind of different applications, being these approached by a wide range of ways.

Assuming possible improvements only about the software and not about the hardware (which could also be treated though), then, it can as modify and improve the algorithms here exposed and explained as add totally new ones:

In the first option it could try to figure out a new approaching for obstacle avoidance with an intermediate level of difficulty, and maybe without so many limitations and restrictions about environment design characteristics. Even it could develop an expansion of this algorithm, by improving the program to able the robot to avoid mobile obstacle, that is, obstacles in motion, that would be a step closer the real life, as the robot could manage people moving in a room, for instance.

Other example might be to try new methods for maze solver,



different ways of which mentioned in this project, as many strategies can be made up and applied.

Regarding adding new algorithms, it could change the reference system, that is, in this project the robot considers itself always as the absolute reference, from its position everything is determined, but it could change to a external reference system by using for example a series of sensors spread and allocated in the environment, with that, it could apply a GPS method for robot positioning and localization.

Or even if being available more sophisticated resources, it could allocate some Bluetooth devices along the environment, whose signals sent to the robot might be clues for it in order to know the position or just meaning instructions for the next steps and so on.

In short, a patrol system is a large field in which can be considered many interesting and diverse applications. A worthy challenge to deep in robotics



BIBLIOGRAPHY

- [1] Aquí las referencias bibliográficas. Con este estilo.
- [CHRI98] JAVA, Teach Yourself. (2001) Author: Chris Wright.
Editorial: McGraw-Hill.
- [JOPE08] Java Foundations. (2008). Introduction to program design and data structures. Authors: John Lewis, Peter DePasquale, Joseph Chase.
Addison Wesley: Pearson International Edition.
- [DAMG07] Lego Mindstorms NXT. (2007). Authors: Dave Astolfo, Mario Ferrari, Giulio Ferrari. Editorial: Syngress
- [LEJOS] <http://lejos.sourceforge.net/index.php>
- [BART08] <http://www.bartneck.de/2008/03/04/java-lego-nxt-eclipse-tutorial/#installJava>



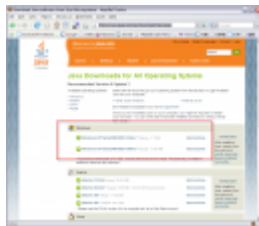
SECTION II USER MANUAL



In this section it is include a kind of tutorial about how to install JAVA and ECLIPSE tool in the computer, in order to set appropriately the environment in which it must work.

Ref [BART08]

You do **NOT** need to install the JDK (Java Developer Kit). We will install Eclipse later, which does already include all the tools you need to write and compile your programs. After the installation you also do **NOT** need to set any Path or Classpath variables for Java itself. Java will be installed in "C:\Program Files\Java". However, if you cannot even get Java to work on your machine, enter the path to your java bin directory into the Path variable.



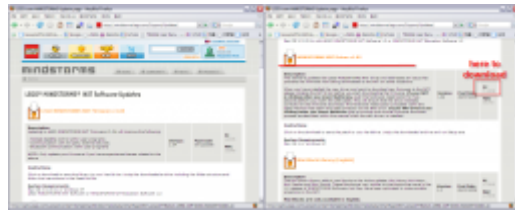
If you already have Java installed you need to verify that you have at least version 5, preferable of the JRE, but the JDK will work as well. We will install Eclipse, which does require at least this version.

5.1 2. Install Lego USB driver on your computer

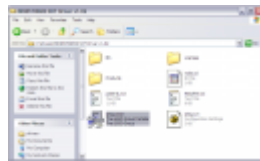
The NXT can be connected to the computer via USB or Bluetooth. The communication via **USB is more reliable** and faster than via Bluetooth. The later also depends on each specific computer model. USB is in comparison more standardized. You must **first install the driver** and only then connect the NXT with the USB cable. You do **NOT** need to install the any software that comes on the Mindstorms CD because we will **NOT** program in NXT-G (Lego's own software that is based on Labview). We only need to install the USB driver which is available on the **Mindstorms' website**. If you already installed the original Mindstorms software then you do not need to uninstall it, just check if the Mindstorms' website has an **update for the USB driver**.



[Download](#) the Mindstorms NXT Driver.



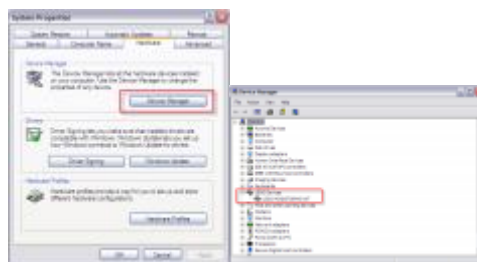
Unzip the file and start setup.exe



Click through the installation program



Windows might ask you to restart the computer after the installation. Next, **connect the NXT** to the computer with the USB cable. Check the correct installation of the driver by checking the device manager. Right-click on "My Computer" and select "Properties". Switch to the hardware tab. Click on "**Device manager**". It should list "Lego Devices => Lego Mindstorms NXT".



5.2.3. Install Lejos on your Computer and on your Mindstorms NXT

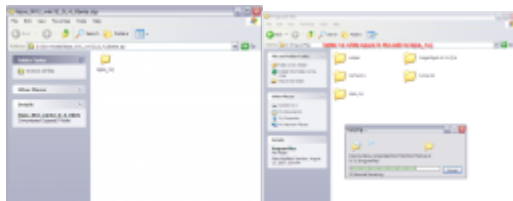
[Download](#) Lejos. At the time this tutorial was created, version Beta 0.4 was the latest release. By now, newer versions are available and we cannot guarantee that this tutorial will work for the newer versions.



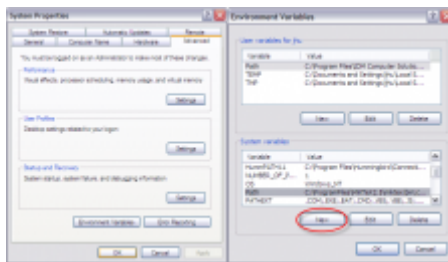


Create a directory called "ProgramFiles" on your hard disk. The already existing directory "Program Files" does contain a **space character** between "Program" and "Files" that sometimes causes **problems** in the usage of Java technology. You will later also install Eclipse into the "ProgramFiles" directory.

Unzip the Lejos ZIP file. This will reveal a directory called "lejos_nxj" . Place it in your "ProgramFiles" directory.



We now need to inform Java of the existence of the Lejos library. Right-click on "My Computer" and select "Properties". Click on the "Advanced" tab. Click on "**Environment Variables**" at the bottom. Create a new "**System variable**" by clicking on "New" at the bottom. Do **NOT** create a "User variable" at the top.



Name the new system variable:

LEJOS_HOME

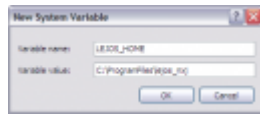
and give it the value:

C:\ProgramFiles\lejos_nxj

In short, Lejos needs to know where it has been installed. Notice that we now already



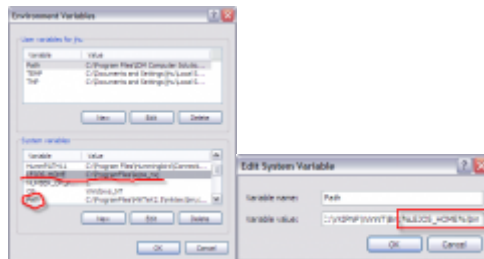
use the new directory "ProgramFiles" that does not contain a space character. When you are done, click "OK".



Verify that the new system variable shows up in the list. Next, we need to add the LEJOS_HOME variable to the system variable "Path". Select "Path" from the list and click on "Edit". Add

```
; %LEJOS_HOME%\bin
```

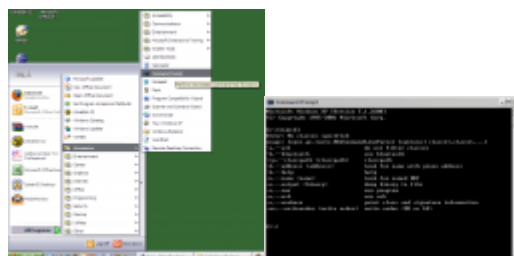
to the end of the value. The different values are separated by a colon and no colon is necessary at the end of the line.



Now we will test if Lejos has been installed correctly by using the DOS command prompt. Click on the Window's "Start" button and then on "All Programs => Accessories => Command Prompt". Type in:

```
lejosdl
```

If the result look like the one in the picture below then everything is fine.



We will now replace the original firmware that came with your NXT with Lejos. You only need to do this once. Lejos will then function as the operating system of your NXT. It implements a Java Virtual Machine on which you can run your Java programs. Make sure that you NXT is connected to your computer with an USB cable and that it has been recognized by



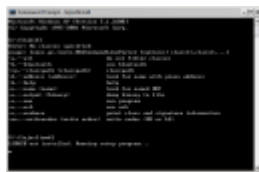
the operating system. In case of doubt check the device manager again. You first need to bring the NXT to the firmware upload mode. Use a paper clip to press and hold the hidden button. The NXT will then play a soft pulsing sound that signals its readiness.



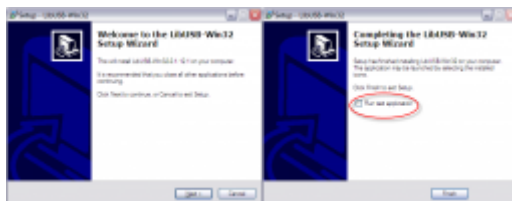
Type:

lejosfirmddl

into the command prompt and press enter.



Lejos requires Libusb to communicate with the NXT. If this free software has not yet been installed on your computer, it will start the Libusb installation wizard. Libusb allows programs to access any USB device on Windows in a generic way. Make sure that you do NOT run the test application in the last dialogue. In case that LeJOS is already installed on your NXT you will still have to install LibUSB to be able to download your programs. You can find it in "lejos_nxj\3rdparty\lib\libusb-win32-filter-bin-0.1.12.1.exe". The name may change when a newer version of libusb becomes available.



Once Libusb is installed the Lejos firmware installer will proceed and hopefully complete the process with success. It should look like the image below.





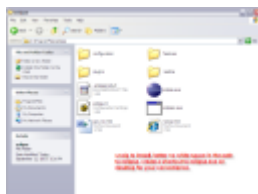
The NXT will now reboot and shortly display the Lejos Logo before changing to the main menu.



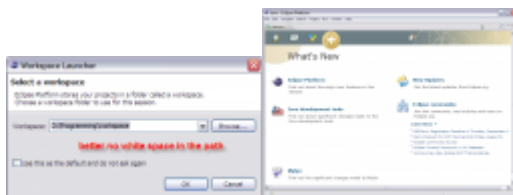
Your NXT is now ready to run Java programs.

5.3 4. Install and configure Eclipse on your Computer

Download Eclipse IDE for Java Developers. Eclipse itself is written in Java and does not contain an installer (setup.exe). It comes in a simple ZIP file that you should unzip into the previously created "ProgramFiles" directory. You may want to create a shortcut to "eclipse.exe" on your desktop for your own convenience.



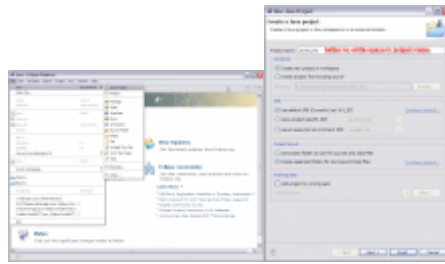
When you start Eclipse you will be asked to choose a workspace. The workspace will contain all the files that you create. Again, you should create a directory in a location so that the path does not contain any space characters " ". In our example we use the directory "d:\programming\workspace". You will then be greeted with the welcome screen.



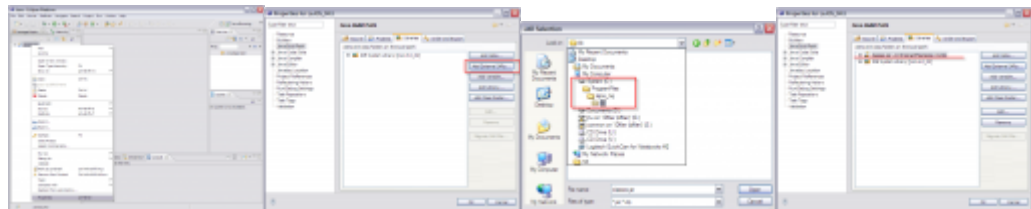
We will now create a Hello World program and upload (others call it download) it into the NXT. First, you need to create a new Java



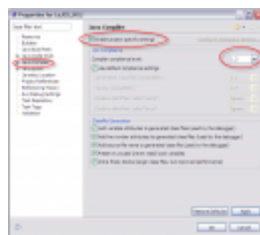
project. Click on "File => New => Java Project". Give the project a new name, but avoid space characters. We called ours "LeJOS NXJ"



Next, you need to turn this project into a Lejos project. Right-click onto the project and select "Properties". Select "Java Build Path" on the left and click on the "Libraries" tab. Afterwards, click on "Add External JARs..." and locate the "lib" directory in your "ProgramFiles\lejos nxj" directory. Select "classes.jar" and press "Open". The Lejos library will now be listed.

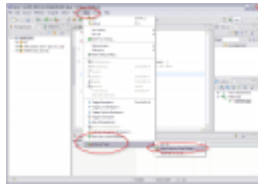


We stay in the properties and move to the "Java Compiler" section listed in the left panel. Check the box "Enable project specific settings" and choose level 1.3 for the compiler compliance level. This will urge the compiler to optimize for an earlier version of Java. This earlier version is more suitable for the NXT since it was developed for embedded systems. It requires much less resources than the latest version of Java. Click on "Apply" and then "OK" to leave the properties.



The next step is to configure Eclipse for downloading the software into the NXT. Click on "Run => External Tools => Open External Tools Dialog..."





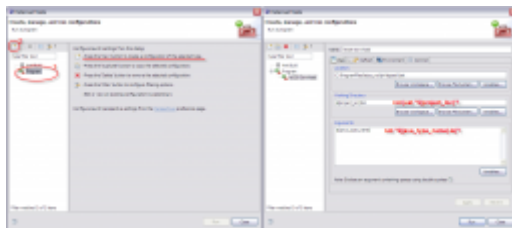
Select "Program" on the left and then click on the "New" icon. Name the tool "leJOS Download" on top. In the "Main" tab, enter the location of the "lejosdl.bat" file that should be in the "lejos-nxi\bin\" directory by clicking on "Browse File System". Enter

`${project_loc}\bin`

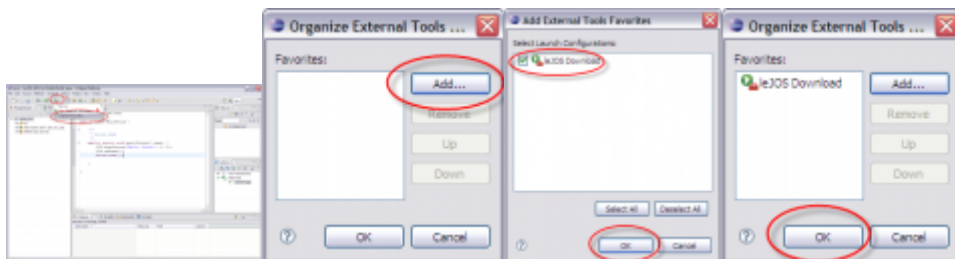
in the "Working directory" and enter

`${java_type_name}`

in the Arguments section.



Now we will create a shortcut for downloading your program into the NXT. Click on the little green "run" icon (with toolbox) and then select "Organize Favorites...". Click on "Add..." in the popup window and then check the box in front of "leJOS Download", which is the external tools we configured in the previous step. Press "OK" and then "Ok" again.

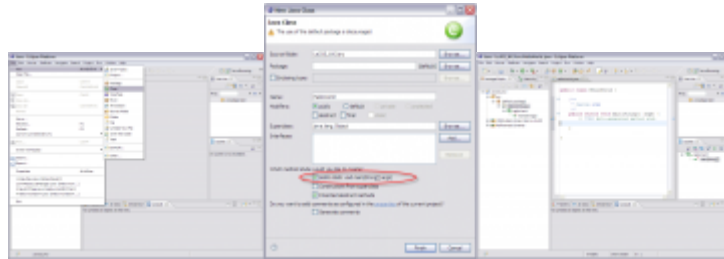


5.4 5. Write, download and run a "Hello world" program

First, we need to create the main class of your program. Click on "New => Class". Name it "HelloWorld" and check the box "public static void(String[] args)". You will receive an almost empty class. Eclipse is a very



smart Java editor that tries to compile your program while you write. Similar to MS Word, it will underline errors with a red line. Sometimes this automatic compilation does not work and you want to force Eclipse to compile. You can achieve this by simply saving the file.



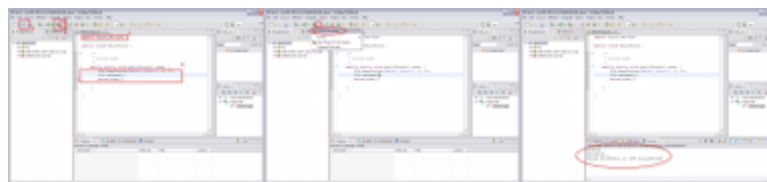
You now need to enter a few lines of Java code. In the first line add:

`import lejos.nxt.LCD;`

This line will enable your program to use the LCD display of your NXT. Next, we need to define what should be displayed and when. Add the following lines to the main method:

```
LCD.drawString("Hello World!", 2, 2);
LCD.refresh();
while(true) {}
```

Now you can save the program and then download it into the NXT. The next screenshot walks you through these steps. Before you start the downloading procedure you must make sure that the NXT is connected to the computer and that the NXT is recognized by the operating system. You can verify this in the device manager. You also need to select your main class window in the middle before starting the downloading procedure, so that Eclipse knows which class you want to download. Use the pull down menu (denoted as 4 in the screenshot) to select the downloading favorite your previously created. This will start the downloading procedure.



After the download is complete you can start the program by using the buttons on the NXT. Notice that the program will run forever. To stop it you need to reset the NXT by holding the two middle buttons for a



longer time. This will then terminate the USB connection, so you need to wait a little for it to reconnect before you download a new program. Of course there are better ways of ending your program, such as listening to a button press or after a certain delay. Replace the endless while loop with this:

```
try _____ {  
Thread.sleep(5000);  
} _____ catch _____ (InterruptedException _____ e) _____ {  
// _____ TODO _____ Auto-generated _____ catch _____ block  
}
```

This will set your NXT to sleep for 5 seconds before the program ends. You will now see "Hello World!" for five seconds and then the program ends. For more information on programming in Java you should check the leJOS page.



SECTION III SOURCE CODE



EVASIVE ROBOT:

- Obstacle
- Avoid
- EvasiveBot
-

+++++

```
import lejos.navigation.*;
import lejos.nxt.*;
```

```
public class Obstacle
{
```

```
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    static UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S1);
    int c = 0;                                //obstacles counter
    int dis = 0;                                // distance to obstacle
    void Closing(){                            //move close to sensed obstacle
        c++;
        Sound.beepSequenceUp();
        do {
            dis = us.getDistance(); //in cm
        }while(dis != 5);
        pilot.stop();                        //when robot is
in 3cm of the obstacle, stops
    }

    int get_dis()
    {
        return dis;
    }
}
```

+++++

```
import java.lang.Math;
import lejos.navigation.Pilot;
import lejos.nxt.Motor;
```

Daniel Egido Sánchez de Vega



```

import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class Avoid
{
    //first try to avoid turning right
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    static UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S1);
    static Obstacle obs = new Obstacle();
    int deg = 0; //angle follower
    static double angle = 0;
    int bound = 0;
    boolean clearway; //true: robot can avoid obstacle by the
right
    int s_long =0;

    //try to avoid the obstacle by the right
    boolean TurnRight()
    {
        do{
            deg = deg - 10;
            pilot.rotate(-10);
            if (us.getDistance() != 255){
                bound = us.getDistance();
                angle = deg;
            }
        }while(deg != -90);
        if (angle < -70){
            clearway = false;
        }else{
            clearway = true;
        }
        return clearway;
    }

    //try to avoid obstacle by the left
    boolean TurnLeft(boolean clearway)
    {
        if (clearway == false){
            deg = 0;
            do{
                deg = deg + 10;
                pilot.rotate(10);
                if (us.getDistance() != 255){
                    bound = us.getDistance();
                    angle = deg;
                }
            }while(deg != 90);
            if (angle > 70){

```



```

        clearway = false;
    }else{
        clearway = true;
    }
    return clearway;
}else{
    return clearway;
}
}

//backwards
void Retry()
{
    while(us.getDistance() != (obs.get_dis() + 5)){
        pilot.backward();
    }
    pilot.stop();
}

//Turn back: Repositioning
void Return()
{
    pilot.rotate(-deg);
}

//Analyze the obstacle
int Analyze()
{
    int travel_long = 0; //distance to
travel
    travel_long = (int) (bound *
Math.sin(Math.toRadians(angle)));
    travel_long = travel_long + 25; //plus robot length
    return travel_long;
}

//Go around the obstacle
void GoAround(int travel_long)
{
    int s_angle = 0;
    int s_bound = 0;
    angle = 0;
    if (clearway == true){
        pilot.travel(travel_long);
        if (deg > 0){
            pilot.rotate(-90);
            pilot.stop();
            while(us.getDistance() != 255){
                angle = angle - 10;
            }
        }
    }
}

```



```

        pilot.rotate(-10);
    }
}
if (deg < 0){
    pilot.rotate(90);
    pilot.stop();
    while(us.getDistance() != 255){
        angle = angle + 10;
        pilot.rotate(10);
    }
}
bound = us.getDistance();
s_angle = (int) angle;
s_long = (int)(s_bound *
Math.sin(Math.toRadians(angle)));
s_long = s_long + 25;
pilot.rotate(-s_angle);
pilot.travel(s_long);
if (deg > 0){
    pilot.rotate(-90);
}
if (deg < 0){
    pilot.rotate(90);
}
pilot.travel(travel_long);
if (deg > 0){
    pilot.rotate(90);
}
if (deg < 0){
    pilot.rotate(-90);
}
pilot.stop();
}
}

int get_s_long()
{
    return s_long;
}

double get_angle()
{
    return angle;
}
}

```

+++++



```

import lejos.navigation.Pilot;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.Sound;
import lejos.nxt.UltrasonicSensor;

public class EvasiveBot
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor
    static UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S1);
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static String dist = "distance: ";
    static String unit = "unit: ";

    public static void main (String[] args)
    {
        int i = 0; //attempts counter
        int n = 0; //avoided obstacles counter
        int waylength = 0; //length of the way to travel
        int total = 0; //total straight distance traveled
        boolean clearway;
        do{
            pilot.resetTachoCount(); //tachometer reset
            pilot.setSpeed(500); //fix speed
            pilot.forward();
            if (us.getDistance() != 255){
                obs.Closing();
                total = total + (int) pilot.getTravelDistance();
                do{
                    clearway = overcome.TurnRight();
                    clearway =
overcome.TurnLeft(clearway);
                    if(clearway == false &&
overcome.get_angle() > 0){
                        overcome.Return();
                        overcome.Retry();
                        i++;
                    }
                    if(clearway == true){
                        waylength = overcome.Analyze();
                        overcome.GoAround(waylength);
                        n++;
                    }
                }while(i != 0 && i < 3);
                if (i > 2){
                    pilot.stop();

```



```
        }
        total = total + overcome.get_s_long();
    }
}
}while(n < 4 || total < 300);
Sound.beepSequence();
LCD.clear();
LCD.drawString(dist,0,0);
LCD.drawInt(total,6,0);
}
}
```



EVASIVE ROBOT 2

- **Obstacle**
- **Avoid**
- **EvasiveBot**

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.*;
import lejos.nxt.UltrasonicSensor;

public class Obstacle
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    int c = 0;           //obstacles counter
    int dis = 0;  // distance to obstacle
    int t = 0;
    void Closing(){           //move close to sensed obstacle
        c++;
        Sound.beepSequenceUp();
        do {
            dis = us1.getDistance(); //in cm
        }while(dis != 5);
        pilot.stop();           //when robot is in 3cm of
the obstacle, stops
    }

    int get_dis()
    {
        return dis;
    }
}

```

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;
import java.lang.Math;

```



```

public class Avoid
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static int deg = 0; //angle follower
    static int d = 0;
    //distance traveled
    static double r_angle = 0; //right bound angle
    static double l_angle = 0; //left bound angle
    static int r_bound = 0; //obstacle's right
bound
    static int l_bound = 0; //obstacle's left
bound
    static int next_bound = 0; //for hole detection
    boolean l_truehole = false; //detection of true
obstacle's left hole
    boolean r_truehole = false; //detection of true
obstacle's right hole
    int l_travel_long = 0; //left distance
to travel
    int r_travel_long = 0; //right distance
to travel
    int r_last_bound = 0;
    int l_last_bound = 0;

    //try to avoid the obstacle by the right
    void TurnRight()
    {
        deg = 0;
        while(us1.getDistance() != 255 || deg != -90 || r_truehole ==
false){
            r_angle = deg;
            deg = deg - 10;
            pilot.rotate(-10);
            r_bound = us1.getDistance();
            overcome.Truebound();
        }
    }

    //try to avoid obstacle by the left
    void TurnLeft()

```



```

{
    deg = 0;
    while(us1.getDistance() != 255 || deg != 90 || l_truehole ==
false){
        l_angle = deg;
        deg = deg + 10;
        pilot.rotate(10);
        l_bound = us1.getDistance();
        overcome.Truebound();
    }
}

//obstacle's hole detection
void Truebound()
{
    int r_hole = 0;
    int l_hole = 0;
    r_hole = ((r_bound+25)/(int)Math.cos(Math.toRadians(90-
deg))) - r_bound;
    l_hole = ((l_bound+25)/(int)Math.cos(Math.toRadians(90-
deg))) - l_bound;
    if (r_hole < (r_bound - r_last_bound))
    {
        r_truehole = true;
    }else{
        r_truehole = false;
    }
    if (l_hole < (l_bound - l_last_bound))
    {
        l_truehole = true;
    }else{
        l_truehole = false;
    }
    r_last_bound = r_bound;
    l_last_bound = l_bound;
}

void SecondaryTruebound()
{
    int sec_r_hole = 0;
    int sec_l_hole = 0;
    int sec_r_bound = 0;
    int sec_l_bound = 0;
    sec_r_bound = us3.getDistance();
    sec_l_bound = us2.getDistance();
    sec_r_hole = ((sec_r_bound-25)/
(int)Math.cos(Math.toRadians(deg))) - sec_r_bound;
    sec_l_hole = ((sec_l_bound-
25)/(int)Math.cos(Math.toRadians(-deg))) - sec_l_bound;
    if (sec_r_hole < (sec_r_bound - r_last_bound))

```



```

    {
        r_truehole = true;
    }else{
        r_truehole = false;
    }
    if (sec_l_hole < (l_bound - l_last_bound))
    {
        l_truehole = true;
    }else{
        l_truehole = false;
    }
    r_last_bound = r_bound;
    l_last_bound = l_bound;
}

//backwards
void Retry()
{
    while(us1.getDistance() != (obs.dis+5)){
        pilot.backward();
    }
    pilot.stop();
}

//Turn back: Repositioning
void Return()
{
    pilot.rotate(-deg);
}

//Analyze the obstacle
void Analyze()
{
    l_travel_long = l_bound *
(int)Math.sin(Math.toRadians(l_angle));
    l_travel_long = l_travel_long + 25;           //plus robot
length
    r_travel_long = r_bound *
(int)Math.sin(Math.toRadians(r_angle));
    r_travel_long = r_travel_long + 25;           //plus robot
length
}

int get_r_travel_long()
{
    return r_travel_long;
}

int get_l_travel_long()
{

```



```

        return l_travel_long;
    }

    int get_l_last_bound()
    {
        return l_last_bound;
    }

    int get_r_last_bound()
    {
        return r_last_bound;
    }

    boolean get_r_truehole()
    {
        return r_truehole;
    }

    boolean get_l_truehole()
    {
        return l_truehole;
    }

    int get_d()
    {
        return d;
    }

    //Go around the obstacle
    void GoAround(int travel_long, boolean rl)
    {
        if (rl == false){
            pilot.rotate(90);
            pilot.travel(travel_long);
            pilot.rotate(-90);
            pilot.resetTachoCount(); //tachometer reset
            pilot.setSpeed(200); //fix speed
            while(us3.getDistance() != 255){
                pilot.forward();
            }
            pilot.travel(25);
            d = (int) pilot.getTravelDistance();
            pilot.rotate(-90);
            pilot.travel(travel_long);
            pilot.rotate(90);
        }
        if (rl == true){
            pilot.rotate(-90);
            pilot.travel(travel_long);
            pilot.rotate(90);
        }
    }

```



```

        pilot.resetTachoCount(); //tachometer reset
        pilot.setSpeed(200); //fix speed
        while(us2.getDistance() != 255){
            pilot.forward();
        }
        pilot.travel(25);
        d = (int) pilot.getTravelDistance();
        pilot.rotate(90);
        pilot.travel(travel_long);
        pilot.rotate(-90);
    }
    pilot.stop();
}
}

+++++

import lejos.navigation.Pilot;
import lejos.nxt.UltrasonicSensor;
import lejos.nxt.*;
import lejos.nxt.LCD;
import lejos.nxt.Motor;

public class EvasiveBot
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static String dist = "distance: ";
    static String unit = "unit: ";

    public static void main (String[] args)
    {
        int i = 0; //attempts counter
        int n = 0; //avoided obstacles

        counter
        boolean r_clearway; //true: robot can avoid
        obstacle by the right side
        boolean l_clearway; //true: robot can avoid
        obstacle by the left side
        int waylength = 0; //length of the way to travel

```



```

        int l_distance = 0;           //distance to left obstacle
        int r_distance = 0;           //distance to right obstacle
        int total = 0;                 //total straight distance

traveled
        boolean rl;                    //true: right shortcut,
false: left shortcut
        do{
            pilot.resetTachoCount(); //tachometer reset
            pilot.setSpeed(500);      //fix speed
            pilot.forward();
            if (us1.getDistance() != 255){
                obs.Closing();
                total = total + (int) pilot.getTravelDistance();
                do{
                    l_distance = us2.getDistance();
                    r_distance = us3.getDistance();
                    if(l_distance == 255){
                        l_clearway = false;
                    }else{
                        l_clearway = true;
                    }
                    if(r_distance == 255){
                        r_clearway = false;
                    }else{
                        r_clearway = true;
                    }
                    overcome.TurnRight();
                    overcome.TurnLeft();
                    overcome.Return();
                    if (l_clearway == true && r_clearway ==
true)
                        {
                            overcome.Analyze();
                            if (overcome.get_r_last_bound()
<= overcome.get_l_last_bound() )
                                {
                                    rl = true;
                                    waylength =
overcome.get_r_travel_long();
                                }else{
                                    rl = false;
                                    waylength =
overcome.get_l_travel_long();
                                }
                                overcome.GoAround(waylength,
rl);
                                n++;
                            }
                            if (l_clearway == true && r_clearway ==
false)

```



```

        {
            if(overcome.get_r_truehole() ==
true)
            {
                overcome.Analyze();
                if
(overcome.get_r_last_bound() <= overcome.get_l_last_bound() )
                {
                    rl = true;
                    waylength =
overcome.get_r_travel_long();
                }else{
                    rl = false;
                    waylength =
overcome.get_l_travel_long();
                }

                overcome.GoAround(waylength, rl);
                n++;
            }else{
                rl = false;
                waylength =
overcome.get_l_travel_long();
                overcome.GoAround(waylength, rl);
                n++;
            }
        }
        if (l_clearway == false && r_clearway ==
true)
        {
            if(overcome.get_l_truehole() ==
true)
            {
                overcome.Analyze();
                if
(overcome.get_r_last_bound() <= overcome.get_l_last_bound() )
                {
                    rl = true;
                    waylength =
overcome.get_r_travel_long();
                }else{
                    rl = false;
                    waylength =
overcome.get_l_travel_long();
                }

                overcome.GoAround(waylength, rl);
                n++;
            }else{

```



```

                                rl = true;
                                waylength =
overcome.get_l_travel_long();

                                overcome.GoAround(waylength, rl);
                                n++;
                                }
                                }
                                if (l_clearway == false && r_clearway ==
false)
                                {
                                    if(overcome.get_l_truehole() ==
true && overcome.get_r_truehole() == true)
                                    {
                                        overcome.Analyze();
                                        if
(overcome.get_r_last_bound() <= overcome.get_l_last_bound() )
                                        {
                                            rl = true;
                                            waylength =
overcome.get_r_travel_long();
                                        }else{
                                            rl = false;
                                            waylength =
overcome.get_l_travel_long();
                                        }

                                        overcome.GoAround(waylength, rl);
                                        n++;
                                    }
                                    if(overcome.get_l_truehole() ==
true && overcome.get_r_truehole() == false)
                                    {
                                        rl = false;
                                        waylength =
overcome.get_l_travel_long();

                                        overcome.GoAround(waylength, rl);
                                        n++;
                                    }
                                    if(overcome.get_l_truehole() ==
false && overcome.get_r_truehole() == true)
                                    {
                                        rl = true;
                                        waylength =
overcome.get_l_travel_long();

                                        overcome.GoAround(waylength, rl);
                                        n++;
                                    }
                                }

```



```

                                if(overcome.get_l_truehole() ==
false && overcome.get_r_truehole() == false)
                                {
                                    overcome.Return();
                                    overcome.Retry();
                                    i++;
                                }
                            }
                        }while(i != 0 && i < 3);
                        if (i > 2){
                            pilot.stop();
                        }
                        total = total + overcome.get_d();
                    }else{
                        total = (int) pilot.getTravelDistance();
                    }
                }while(n < 4 || total < 300);
                Sound.beepSequence();
                LCD.clear();
                LCD.drawString(dist,0,0);
                LCD.drawInt(total,6,0);
            }
        }
    }

```



EASY AVOIDANCE

- Sensing
- Actuator
- Obstacle (idem)
- Avoid
- EvasiveBot

+++++

```
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;
```

```
public class Sensing
{
    static UltrasonicSensor us = new
    UltrasonicSensor(SensorPort.S1);

    int distance(UltrasonicSensor us)
    {
        return us.getDistance();
    }
}
```

+++++

```
import lejos.navigation.Pilot;
import lejos.nxt.Motor;
```

```
public class Actuator
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor

    void go()
    {
        pilot.forward();
    }

    void back()
```



```

    {
        pilot.backward();
    }

    int distance()
    {
        return (int) pilot.getTravelDistance();
    }

    void counter()
    {
        pilot.resetTachoCount();
    }

    void rotate(int angle)
    {
        pilot.rotate(angle);
    }

    void speed(int speed)
    {
        pilot.setSpeed(speed);
    }

    void travel(int d)
    {
        pilot.travel(d);
    }

    void stop()
    {
        pilot.stop();
    }
}

+++++

import lejos.navigation.Pilot;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class Avoid
{
    //first try to avoid turning right
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    static UltrasonicSensor us = new
    UltrasonicSensor(SensorPort.S1);
    static Obstacle obs = new Obstacle();

```



```

static Actuator actuator = new Actuator();
static Sensing sensing = new Sensing();
boolean clearway;           //true: robot can avoid obstacle by the
right
int l1 = 0;
int l2 = 0;

//try to avoid the obstacle by the right
boolean TurnRight()
{
    actuator.rotate(-90);
    if (sensing.distance(us) != 255){
        clearway = false;
    }else{
        clearway = true;
    }
    return clearway;
}

//try to avoid obstacle by the left
boolean TurnLeft()
{
    actuator.rotate(90);
    if (sensing.distance(us) != 255){
        clearway = false;
    }else{
        clearway = true;
    }
    return clearway;
}

//backwards
void Retry()
{
    while(sensing.distance(us) != (obs.get_dis() + 5)){
        actuator.back();
    }
    actuator.stop();
}

//Turn back: Repositioning
void Return()
{
    actuator.rotate(-90);
}

//Go around the obstacle
void RAround()
{
    do{

```



```

        actuator.rotate(-90);
        actuator.travel(40);
        l1 += 40;
        actuator.rotate(90);
        pilot.stop();
    }while(sensing.distance(us) != 255);
    actuator.travel(40);
    l2 += 40;
    actuator.rotate(90);
    while(sensing.distance(us) != 255){
        actuator.rotate(-90);
        actuator.travel(30);
        l2 += 30;
        actuator.rotate(90);
    }
    actuator.travel(l1);
    actuator.rotate(-90);
}

void LAround()
{
    do{
        actuator.rotate(90);
        actuator.travel(40);
        l1 += 40;
        actuator.rotate(-90);
        pilot.stop();
    }while(sensing.distance(us) != 255);
    actuator.travel(40);
    l2 += 40;
    actuator.rotate(-90);
    while(sensing.distance(us) != 255){
        actuator.rotate(90);
        actuator.travel(30);
        l2 += 30;
        actuator.rotate(-90);
    }
    actuator.travel(l1);
    actuator.rotate(90);
}

int get_l1()
{
    return l1;
}

int get_l2()
{
    return l1;
}

```



```
}
```

```
+++++
```

```
import lejos.navigation.Pilot;
import lejos.nxt.LCD;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.Sound;
import lejos.nxt.UltrasonicSensor;

public class EvasiveBot
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor
    static UltrasonicSensor us = new
UltrasonicSensor(SensorPort.S1);
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static String dist = "distance: ";
    static String unit = "unit: ";

    public static void main (String[] args)
    {
        int i = 0; //attempts counter
        int n = 0; //avoided obstacles counter
        int total = 0; //total straight distance traveled
        boolean clearway;
        do{
            actuator.counter(); //tachometer reset
            actuator.speed(500); //fix speed
            actuator.go();
            if (sensing.distance(us) != 255){
                obs.Closing();
                total = total + actuator.distance();
                do{
                    clearway = overcome.TurnRight();
                    if (clearway == true)
                    {
                        actuator.rotate(90);
                        overcome.RAround();
                        n++;
                    }else{
                        clearway = overcome.TurnLeft();
                        if (clearway == true)
                        {
```



```
        actuator.rotate(-90);
        overcome.LAround();
        n++;
    }else{
        overcome.Return();
        overcome.Retry();
        i++;
    }
}
}while(i != 0 && i < 3);
if (i > 2){
    actuator.stop();
}
total = total + overcome.get_l2();
}
}while(n < 4 || total < 300);
Sound.beepSequence();
LCD.clear();
LCD.drawString(dist,0,0);
LCD.drawInt(total,6,0);
}
}
```



FROM A TO B: STRAIGHT

- **Obstacle (Idem)**
- **Avoid (Idem)**
- **EvasiveBot**
- **AB_STRAIGHT**

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.UltrasonicSensor;
import lejos.nxt.*;
import lejos.nxt.LCD;
import lejos.nxt.Motor;

public class EvasiveBot
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static String dist = "distance: ";
    static String unit = "unit: ";
    int total = 0;                                //total straight distance traveled

    void Evasion()
    {
        int i = 0;                                //attempts counter
        int n = 0;                                //avoided obstacles
counter
        boolean r_clearway;                        //true: robot can avoid
obstacle by the right side
        boolean l_clearway;                        //true: robot can avoid
obstacle by the left side
        int waylength = 0;                        //length of the way to travel

```



```

    int l_distance = 0;           //distance to left obstacle
    int r_distance = 0;           //distance to right obstacle
    boolean rl;                   //true: right shortcut,
false: left shortcut
    //do{
        pilot.resetTachoCount(); //tachometer reset
        pilot.setSpeed(500);      //fix speed
        pilot.forward();
        if (us1.getDistance() != 255){
            obs.Closing();
            total = total + (int) pilot.getTravelDistance();
            do{
                l_distance = us2.getDistance();
                r_distance = us3.getDistance();
                if(l_distance == 255){
                    l_clearway = false;
                }else{
                    l_clearway = true;
                }
                if(r_distance == 255){
                    r_clearway = false;
                }else{
                    r_clearway = true;
                }
                overcome.TurnRight();
                overcome.TurnLeft();
                overcome.Return();
                if (l_clearway == true && r_clearway ==
true)
                {
                    overcome.Analyze();
                    if (overcome.get_r_last_bound()
<= overcome.get_l_last_bound() )
                    {
                        rl = true;
                        waylength =
overcome.get_r_travel_long();
                    }else{
                        rl = false;
                        waylength =
overcome.get_l_travel_long();
                    }
                    overcome.GoAround(waylength,
rl);
                    n++;
                    i = 0;
                }
                if (l_clearway == true && r_clearway ==
false)
                {

```



```

true)
    if(overcome.get_r_truehole() ==
    {
        overcome.Analyze();
        if
(overcome.get_r_last_bound() <= overcome.get_l_last_bound() )
        {
            rl = true;
            waylength =
overcome.get_r_travel_long();
        }else{
            rl = false;
            waylength =
overcome.get_l_travel_long();
        }

        overcome.GoAround(waylength, rl);
        n++;
    }else{
        rl = false;
        waylength =
overcome.get_l_travel_long();
        overcome.GoAround(waylength, rl);
        n++;
    }
    i = 0;
}
if (l_clearway == false && r_clearway ==
true)
{
    if(overcome.get_l_truehole() ==
true)
    {
        overcome.Analyze();
        if
(overcome.get_r_last_bound() <= overcome.get_l_last_bound() )
        {
            rl = true;
            waylength =
overcome.get_r_travel_long();
        }else{
            rl = false;
            waylength =
overcome.get_l_travel_long();
        }

        overcome.GoAround(waylength, rl);
        n++;
    }else{

```



```

                                rl = true;
                                waylength =
overcome.get_l_travel_long();

                                overcome.GoAround(waylength, rl);
                                n++;
                                }
                                i = 0;
                                }
                                if (l_clearway == false && r_clearway ==
false)
                                {
                                if(overcome.get_l_truehole() ==
true && overcome.get_r_truehole() == true)
                                {
                                overcome.Analyze();
                                if
(overcome.get_r_last_bound() <= overcome.get_l_last_bound() )
                                {
                                rl = true;
                                waylength =
overcome.get_r_travel_long();
                                }else{
                                rl = false;
                                waylength =
overcome.get_l_travel_long();
                                }

                                overcome.GoAround(waylength, rl);
                                n++;
                                i = 0;
                                }
                                if(overcome.get_l_truehole() ==
true && overcome.get_r_truehole() == false)
                                {
                                rl = false;
                                waylength =
overcome.get_l_travel_long();
                                overcome.GoAround(waylength, rl);
                                n++;
                                i = 0;
                                }
                                if(overcome.get_l_truehole() ==
false && overcome.get_r_truehole() == true)
                                {
                                rl = true;
                                waylength =
overcome.get_l_travel_long();

```



```

        overcome.GoAround(waylength, rl);
                                n++;
                                i = 0;
                                }
                                if(overcome.get_l_truehole() ==
false && overcome.get_r_truehole() == false)
                                {
                                        overcome.Return();
                                        overcome.Retry();
                                        i++;
                                }
                                }
                                }while(i != 0 && i < 3);
                                if (i > 2){
                                        pilot.stop();
                                }
                                total = total + overcome.get_d();
                                }else{
                                        total = (int) pilot.getTravelDistance();
                                }
                                }
                                //}while(n < 4 || total < 300);
                                Sound.beepSequence();
                                LCD.clear();
                                LCD.drawString(dist,0,0);
                                LCD.drawInt(total,6,0);
                                }

                                //total distance travelled
                                int get_total()
                                {
                                        return total;
                                }
                                }

```

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.*;

import java.lang.Math;

public class AB_STRAIGHT
{
    static AB_STRAIGHT voyager = new AB_STRAIGHT();
    static EvasiveBot wayfinder = new EvasiveBot();
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();

```



```

    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);

    void Orientation(int x2, int y2)
    {
        //Firstly it is assumed the initial robot position as zero and
north-oriented
        int x1 = 0; //initial x position
        int y1 = 0; //initial y position
        int LookAt = 0; //angle to turn to ahead
point B
        LookAt = (int)Math.toDegrees(Math.atan((y2-y1)/(x2-x1)));
        pilot.rotate(LookAt);
    }

    void keepLdistance()
    {
        pilot.forward();
        pilot.setSpeed(200);
        while(us2.getDistance() > 10 && us3.getDistance() < 25)
        {
            Motor.B.setSpeed(210);
        }
        pilot.setSpeed(200);
    }

    void keepRdistance()
    {
        pilot.forward();
        pilot.setSpeed(200);
        while(us3.getDistance() > 10 && us3.getDistance() < 25)
        {
            Motor.A.setSpeed(210);
        }
        pilot.setSpeed(200);
    }

    void WayCorrection()
    {
        int d1 = 0;
        int d2 = 0;
        int d3 = 0;
        if (overcome.get_part() == 1)
        {

```



```

        pilot.resetTachoCount();
        while(us3.getDistance() < 25)
        {
            voyager.keepRdistance();
        }
        d1 = (int) pilot.getTravelDistance();
        pilot.rotate(-90);
        pilot.resetTachoCount();
        while(us3.getDistance() < 25)
        {
            voyager.keepRdistance();
        }
        d2 = (int) pilot.getTravelDistance();
        if (d2 > overcome.get_l_travel_long())
        {
            pilot.rotate(-90);
            pilot.resetTachoCount();
            while(us3.getDistance() < 25)
            {
                voyager.keepRdistance();
            }
            d3 = (int) pilot.getTravelDistance();
            if (d3 > 2*d1)
            {
                pilot.stop();
                LCD.clear();
                LCD.drawString("no way",0,0);
            }else{
                pilot.rotate(-90);
                pilot.travel(d2 -
overcome.get_l_travel_long());
                pilot.rotate(90);
                pilot.travel(d1 - d3);
                pilot.stop();
                Sound.beepSequence();
                LCD.clear();
                LCD.drawString("Destination",0,0);
            }
        }else{
            pilot.rotate(-90);
            pilot.travel(d1);
            pilot.rotate(90);
            pilot.travel(overcome.get_l_travel_long() - d2);
            pilot.stop();
            Sound.beepSequence();
            LCD.clear();
            LCD.drawString("Destination",0,0);
        }
    }
    if (overcome.get_part() == 2)

```



```

{
    pilot.stop();
    Sound.beepSequence();
    LCD.clear();
    LCD.drawString("Destination",0,0);
}
if (overcome.get_part() == 3)
{
    pilot.resetTachoCount();
    while(us2.getDistance() < 25)
    {
        voyager.keepLdistance();
    }
    d1 = (int) pilot.getTravelDistance();
    pilot.rotate(90);
    pilot.resetTachoCount();
    while(us2.getDistance() < 25)
    {
        voyager.keepLdistance();
    }
    d2 = (int) pilot.getTravelDistance();
    if (d2 > overcome.get_r_travel_long())
    {
        pilot.rotate(90);
        pilot.resetTachoCount();
        while(us2.getDistance() < 25)
        {
            voyager.keepLdistance();
        }
        d3 = (int) pilot.getTravelDistance();
        if (d3 > 2*d1)
        {
            pilot.stop();
            LCD.clear();
            LCD.drawString("no way",0,0);
        }else{
            pilot.rotate(90);
            pilot.travel(d2 -
overcome.get_r_travel_long());
            pilot.rotate(-90);
            pilot.travel(d1 - d3);
            pilot.stop();
            Sound.beepSequence();
            LCD.clear();
            LCD.drawString("Destination",0,0);
        }
    }else{
        pilot.rotate(90);
        pilot.travel(d1);
        pilot.rotate(-90);
    }
}

```



```

        pilot.travel(overcome.get_r_travel_long() - d2);
        pilot.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}
if (overcome.get_part() == 4)
{
    pilot.stop();
    Sound.beepSequence();
    LCD.clear();
    LCD.drawString("Destination",0,0);
}
}

public static void main (String[] args)
{
    //Firstly it has to provide the destination
    int x2 = 150;
    int y2 = 150;
    voyager.Orientation(x2, y2);
    do{
        wayfinder.Evasion();
    }while(wayfinder.get_total() <
Math.sqrt(Math.pow(x2,2)+Math.pow(y2,2)) && overcome.get_part() == 0);
    if (wayfinder.get_total() ==
Math.sqrt(Math.pow(x2,2)+Math.pow(y2,2)))
    {
        if (overcome.get_part() != 0)
        {
            voyager.WayCorrection();
        }
        if (overcome.get_part() == 0)
        {
            pilot.stop();
            Sound.beepSequence();
            LCD.clear();
            LCD.drawString("Destination",0,0);
        }
    }
}
}
}

```



AB: THE LEAST X

- Sensing (idem)
- Actuator (idem)
- Obstacle (idem)
- Avoid
- EvasiveBot
- AB_LeastX

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class Avoid
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();
    static int d = 0;
    //distance traveled
    int part = 0;                                //to
    identify the obstacle's stretch
    int l1 = 0;
    int l2 = 0;

    //backwards
    void Retry()
    {
        while(us1.getDistance() != (obs.dis+5)){
            pilot.backward();

```



```

    }
    pilot.stop();
}

//Turn back: Repositioning
void Return()
{
    actuator.rotate(-90);
}

void RAround()
{
    actuator.rotate(-90);
    actuator.counter();
    do{
        actuator.go();
        part = 1;
    }while(sensing.distance(us2) != 255);
    l1 = actuator.distance();
    actuator.rotate(90);
    actuator.counter();
    while(sensing.distance(us2) != 255){
        actuator.go();
        part = 2;
    }
    l2 = actuator.distance();
    actuator.rotate(90);
    actuator.travel(l1);
    actuator.rotate(-90);
}

void LAround()
{
    actuator.rotate(90);
    actuator.counter();
    do{
        actuator.go();
        part = 3;
    }while(sensing.distance(us3) != 255);
    l1 = actuator.distance();
    actuator.rotate(-90);
    actuator.counter();
    actuator.go();
    while(sensing.distance(us3) != 255){
        actuator.go();
        part = 4;
    }
    l2 = actuator.distance();
    actuator.rotate(-90);
    actuator.travel(l1);
}

```



```

        actuator.rotate(90);
    }

    int get_l1()
    {
        return l1;
    }

    int get_l2()
    {
        return l1;
    }

    int get_d()
    {
        return d;
    }

    int get_part()
    {
        return part;
    }
}

+++++

import lejos.navigation.Pilot;
import lejos.nxt.*;

public class EvasiveBot
{
    static Pilot pilot = new Pilot(5.6f,10.5f,Motor.A,Motor.C);
    //wheel diameter, trackwidth, leftmotor, rightmotor
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();
    static String dist = "distance: ";
    static String unit = "unit: ";
    int total = 0;                                //total straight distance traveled

```



```

void Evasion()
{
    int i = 0;                                //attempts counter
    int n = 0;                                //avoided obstacles

    counter
    boolean r_clearway;                        //true: robot can avoid
    obstacle by the right side
    boolean l_clearway;                        //true: robot can avoid
    obstacle by the left side
    int l_distance = 0;                        //distance to left obstacle
    int r_distance = 0;                        //distance to right obstacle
    //do{
        actuator.counter(); //tachometer reset
        actuator.speed(500);                    //fix speed
        actuator.go();
        if (us1.getDistance() != 255){
            obs.Closing();
            total = total + (int) actuator.distance();
            do{
                l_distance = sensing.distance(us2);
                r_distance = sensing.distance(us3);
                if(l_distance == 255){
                    l_clearway = false;
                }else{
                    l_clearway = true;
                }
                if(r_distance == 255){
                    r_clearway = false;
                }else{
                    r_clearway = true;
                }

                if (l_clearway == true && r_clearway ==
true)
                {
                    overcome.RAround();
                    n++;
                    i = 0;
                }
                if (l_clearway == true && r_clearway ==
false)
                {
                    overcome.LAround();
                    n++;
                    i = 0;
                }
                if (l_clearway == false && r_clearway ==
true)
                {
                    overcome.RAround();

```



```

        n++;
        i = 0;
    }
    if (l_clearway == false && r_clearway ==
false)
    {
        overcome.Retry();
        n++;
        i ++;
    }
    while(i != 0 && i < 3);
    if (i > 2){
        actuator.stop();
    }
    total = total + overcome.get_l2();
} else{
    total = (int) actuator.distance();
}
//}while(n < 4 || total < 300);
Sound.beepSequence();
LCD.clear();
LCD.drawString(dist,0,0);
LCD.drawInt(total,6,0);
}

//total distance travelled
int get_total()
{
    return total;
}
}

```

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.*;

```

```

public class AB_leastX
{
    static AB_leastX voyager = new AB_leastX();
    static EvasiveBot wayfinder = new EvasiveBot();
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);

```



```

    static UltrasonicSensor us2 = new
    UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
    UltrasonicSensor(SensorPort.S3);
    //Firstly it is assumed the initial robot position as zero and north-
    oriented
    static int x1 = 0;           //initial x position
    static int y1 = 0;           //initial y position

    void LeastX(int x2, int y2)
    {
        if (x2 > x1)
        {
            actuator.rotate(-90);
        } else if (x2 < x1)
        {
            actuator.rotate(90);
        }
    }

    void keepLdistance()
    {
        actuator.go();
        actuator.speed(200);
        while(sensing.distance(us2) > 10 && sensing.distance(us2)
< 25)
        {
            Motor.B.setSpeed(210);
        }
        actuator.speed(200);
    }

    void keepRdistance()
    {
        actuator.go();
        actuator.speed(200);
        while(sensing.distance(us3) > 10 && sensing.distance(us3)
< 25)
        {
            Motor.A.setSpeed(210);
        }
        actuator.speed(200);
    }

    void WayCorrection()
    {
        int d1 = 0;
        int d2 = 0;
        int d3 = 0;
        if (overcome.get_part() == 1)

```



```

{
    actuator.counter();
    while(sensing.distance(us3) < 25)
    {
        voyager.keepRdistance();
    }
    d1 = actuator.distance();
    actuator.rotate(-90);
    actuator.counter();
    while(sensing.distance(us3) < 25)
    {
        voyager.keepRdistance();
    }
    d2 = actuator.distance();
    if (d2 > overcome.get_l1())
    {
        actuator.rotate(-90);
        actuator.counter();
        while(sensing.distance(us3) < 25)
        {
            voyager.keepRdistance();
        }
        d3 = actuator.distance();
        if (d3 > 2*d1)
        {
            actuator.stop();
            LCD.clear();
            LCD.drawString("no way",0,0);
        }else{
            actuator.rotate(-90);
            actuator.travel(d2 - overcome.get_l1());
            actuator.rotate(90);
            actuator.travel(d1 - d3);
            actuator.stop();
            Sound.beepSequence();
            LCD.clear();
            LCD.drawString("Destination",0,0);
        }
    }else{
        actuator.rotate(-90);
        actuator.travel(d1);
        actuator.rotate(90);
        actuator.travel(overcome.get_l1() - d2);
        actuator.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}
if (overcome.get_part() == 2)

```



```

{
    actuator.stop();
    Sound.beepSequence();
    LCD.clear();
    LCD.drawString("Destination",0,0);
}
if (overcome.get_part() == 3)
{
    actuator.counter();
    while(sensing.distance(us2) < 25)
    {
        voyager.keepLdistance();
    }
    d1 = actuator.distance();
    actuator.rotate(90);
    actuator.counter();
    while(sensing.distance(us2) < 25)
    {
        voyager.keepLdistance();
    }
    d2 = actuator.distance();
    if (d2 > overcome.get_l1())
    {
        actuator.rotate(90);
        actuator.counter();
        while(sensing.distance(us2) < 25)
        {
            voyager.keepLdistance();
        }
        d3 = actuator.distance();
        if (d3 > 2*d1)
        {
            actuator.stop();
            LCD.clear();
            LCD.drawString("no way",0,0);
        }else{
            actuator.rotate(90);
            actuator.travel(d2 - overcome.get_l1());
            actuator.rotate(-90);
            actuator.travel(d1 - d3);
            actuator.stop();
            Sound.beepSequence();
            LCD.clear();
            LCD.drawString("Destination",0,0);
        }
    }else{
        actuator.rotate(90);
        actuator.travel(d1);
        actuator.rotate(-90);
    }
}

```



```

        actuator.travel(overcome.get_l1() - d2);
        actuator.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}
if (overcome.get_part() == 4)
{
    actuator.stop();
    Sound.beepSequence();
    LCD.clear();
    LCD.drawString("Destination",0,0);
}
}

public static void main (String[] args)
{
    //Firstly it has to provide the destination
    int x2 = 150;
    int y2 = 150;
    voyager.LeastX(x2, y2);
    do{
        wayfinder.Evasion();
    }while(wayfinder.get_total() < x2);
    if(y2 > y1)
    {
        actuator.rotate(-90);
        do{
            wayfinder.Evasion();
        }while(wayfinder.get_total() != (x2 + y2));
    }else if(y2 < y1){
        actuator.rotate(90);
        do{
            wayfinder.Evasion();
        }while(wayfinder.get_total() != (x2 + y2));
        if (wayfinder.get_total() == (x2 + y2))
        {
            if (overcome.get_part() != 0)
            {
                voyager.WayCorrection();
            }
            if (overcome.get_part() == 0)
            {
                actuator.stop();
                Sound.beepSequence();
                LCD.clear();
                LCD.drawString("Destination",0,0);
            }
        }
    }
}

```



```
        }else{  
            actuator.stop();  
            Sound.beepSequence();  
            LCD.clear();  
            LCD.drawString("Destination",0,0);  
        }  
    }  
}
```



AB: THE LEAST Y

- Sensing (idem)
- Actuator (idem)
- Obstacle (idem)
- Avoid (idem)
- EvasiveBot(idem)
- AB_LeastY

+++++

```
import lejos.navigation.Pilot;
import lejos.nxt.*;
```

```
public class AB_leastY
{
```

```
    static AB_leastY voyager = new AB_leastY();
    static EvasiveBot wayfinder = new EvasiveBot();
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    //Firstly it is assumed the initial robot position as zero and north-
oriented
    static int x1 = 0; //initial x position
    static int y1 = 0; //initial y position

    void LeastY(int x2, int y2)
    {
        if (y2 < y1)
        {
            actuator.rotate(180);
        }
    }
}
```



```

void keepLdistance()
{
    actuator.go();
    actuator.speed(200);
    while(sensing.distance(us2) > 10 && sensing.distance(us2)
< 25)
    {
        Motor.B.setSpeed(210);
    }
    actuator.speed(200);
}

void keepRdistance()
{
    actuator.go();
    actuator.speed(200);
    while(sensing.distance(us3) > 10 && sensing.distance(us3)
< 25)
    {
        Motor.A.setSpeed(210);
    }
    actuator.speed(200);
}

void WayCorrection()
{
    int d1 = 0;
    int d2 = 0;
    int d3 = 0;
    if (overcome.get_part() == 1)
    {
        actuator.counter();
        while(sensing.distance(us3) < 25)
        {
            voyager.keepRdistance();
        }
        d1 = actuator.distance();
        actuator.rotate(-90);
        actuator.counter();
        while(sensing.distance(us3) < 25)
        {
            voyager.keepRdistance();
        }
        d2 = actuator.distance();
        if (d2 > overcome.get_l1())
        {
            actuator.rotate(-90);
            actuator.counter();
            while(sensing.distance(us3) < 25)
            {

```



```

        voyager.keepRdistance();
    }
    d3 = actuator.distance();
    if (d3 > 2*d1)
    {
        actuator.stop();
        LCD.clear();
        LCD.drawString("no way",0,0);
    }else{
        actuator.rotate(-90);
        actuator.travel(d2 - overcome.get_l1());
        actuator.rotate(90);
        actuator.travel(d1 - d3);
        actuator.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}
}
}
else{
    actuator.rotate(-90);
    actuator.travel(d1);
    actuator.rotate(90);
    actuator.travel(overcome.get_l1() - d2);
    actuator.stop();
    Sound.beepSequence();
    LCD.clear();
    LCD.drawString("Destination",0,0);
}
}
}
if (overcome.get_part() == 2)
{
    actuator.stop();
    Sound.beepSequence();
    LCD.clear();
    LCD.drawString("Destination",0,0);
}
}
if (overcome.get_part() == 3)
{
    actuator.counter();
    while(sensing.distance(us2) < 25)
    {
        voyager.keepLdistance();
    }
    d1 = actuator.distance();
    actuator.rotate(90);
    actuator.counter();
    while(sensing.distance(us2) < 25)
    {
        voyager.keepLdistance();
    }
}
}

```



```

        d2 = actuator.distance();
        if (d2 > overcome.get_l1())
        {
            actuator.rotate(90);
            actuator.counter();
            while(sensing.distance(us2) < 25)
            {
                voyager.keepLdistance();
            }
            d3 = actuator.distance();
            if (d3 > 2*d1)
            {
                actuator.stop();
                LCD.clear();
                LCD.drawString("no way",0,0);
            }else{
                actuator.rotate(90);
                actuator.travel(d2 - overcome.get_l1());
                actuator.rotate(-90);
                actuator.travel(d1 - d3);
                actuator.stop();
                Sound.beepSequence();
                LCD.clear();
                LCD.drawString("Destination",0,0);
            }
        }else{
            actuator.rotate(90);
            actuator.travel(d1);
            actuator.rotate(-90);
            actuator.travel(overcome.get_l1() - d2);
            actuator.stop();
            Sound.beepSequence();
            LCD.clear();
            LCD.drawString("Destination",0,0);
        }
    }
    if (overcome.get_part() == 4)
    {
        actuator.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}

public static void main (String[] args)
{
    //Firstly it has to provide the destination
    int x2 = 150;
    int y2 = 150;

```



```

voyager.LeastY(x2, y2);
if(y2 == y1)
{
    if (x2 > x1)
    {
        actuator.rotate(-90);
        do{
            wayfinder.Evasion();
        }while(wayfinder.get_total() < x2);
    }else if(x2 < x1){
        actuator.rotate(90);
        do{
            wayfinder.Evasion();
        }while(wayfinder.get_total() < x2);
    }else{
        actuator.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}
else if(y2 != y1){
    do{
        wayfinder.Evasion();
    }while(wayfinder.get_total() != y2);
    if (x2 > x1)
    {
        actuator.rotate(-90);
        do{
            wayfinder.Evasion();
        }while(wayfinder.get_total() < (y2 + x2));
    }else if(x2 < x1){
        actuator.rotate(90);
        do{
            wayfinder.Evasion();
        }while(wayfinder.get_total() < (y2 + x2));
    }else{
        actuator.stop();
        Sound.beepSequence();
        LCD.clear();
        LCD.drawString("Destination",0,0);
    }
}
if (wayfinder.get_total() == (x2 + y2))
{
    if (overcome.get_part() != 0)
    {
        voyager.WayCorrection();
    }
    if (overcome.get_part() == 0)

```



```
        {  
            actuator.stop();  
            Sound.beepSequence();  
            LCD.clear();  
            LCD.drawString("Destination",0,0);  
        }  
    }  
}
```



HUNTER

- Sensing (Idem)
- Actuator (Idem)
- Obstacle (Idem)
- Avoid (Idem)
- EvasiveBot (Idem)
- Hunter
-

+++++

```

import lejos.navigation.Pilot;
import lejos.nxt.*;

public class Hunter
{
    static Hunter predator = new Hunter();
    static EvasiveBot wayfinder = new EvasiveBot();
    static Obstacle obs = new Obstacle();
    static Avoid overcome = new Avoid();
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();
    static UltrasonicSensor us1 = new
UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
UltrasonicSensor(SensorPort.S3);
    static LightSensor ls = new LightSensor(SensorPort.S4);
    //Firstly it is assumed the initial robot position as zero and north-
    oriented
    static int x1 = 0; //initial x position
    static int y1 = 0; //initial y position

    void LeastX(int x2, int y2)
    {
        if (x2 > x1)
        {
            actuator.rotate(-90);
        } else if (x2 < x1)
        {
            actuator.rotate(90);
        }
    }

    void keepLdistance()
    {

```



```

        actuator.go();
        actuator.speed(200);
        while(sensing.distance(us2) > 10 && sensing.distance(us2)
< 25)
        {
                Motor.B.setSpeed(210);
        }
        actuator.speed(200);
    }

    void keepRdistance()
    {
        actuator.go();
        actuator.speed(200);
        while(sensing.distance(us3) > 10 && sensing.distance(us3)
< 25)
        {
                Motor.A.setSpeed(210);
        }
        actuator.speed(200);
    }

    //prey detection
    static void det()
    {
        if(ls.readValue() > 50)
        {
                Hunter.Hunting();
        }
    }

    //Prowling
    static void Zigzag()
    {
        while(ls.readValue() < 50)
        {
                do{
                        wayfinder.Evasion();
                        Hunter.det();
                }while(wayfinder.get_total() < 80);
                actuator.rotate(135);
                do{
                        wayfinder.Evasion();
                        Hunter.det();
                }while(wayfinder.get_total() < 80);
                actuator.rotate(360);
                do{
                        wayfinder.Evasion();
                        Hunter.det();
                }while(wayfinder.get_total() < 80);
        }
    }

```



```

        actuator.rotate(-135);
        actuator.travel(30);
        Hunter.det();
        actuator.rotate(-45);
        do{
            wayfinder.Evasion();
            Hunter.det();
        }while(wayfinder.get_total() < 60);
        actuator.rotate(360);
        do{
            wayfinder.Evasion();
            Hunter.det();
        }while(wayfinder.get_total() < 80);
        actuator.rotate(135);
        do{
            wayfinder.Evasion();
            Hunter.det();
        }while(wayfinder.get_total() < 80);
        actuator.rotate(360);
        do{
            wayfinder.Evasion();
            Hunter.det();
        }while(wayfinder.get_total() < 80);
        actuator.travel(133);
        Hunter.det();
        actuator.rotate(180);
    }
}

//searching
static void searching()
{
    //turn right to search
    actuator.rotate(-90);
    if(Is.readValue() > 50)
    {
        Hunter.Hunting();
        Hunter.searching();
    }
    //turn left to search
    actuator.rotate(90);
    if(Is.readValue() > 50)
    {
        Hunter.Hunting();
        Hunter.searching();
    }
    actuator.stop();           //Hunter lost the prey
}

static void Hunting()

```



```

    {
        while(ls.readValue() > 50)
        {
            actuator.go();
            if (ls.readValue() > 85)
            {
                actuator.stop();    //Hunter wins
            }
        }
        Hunter.searching();
    }

    public static void main (String[] args)
    {
        //Firstly it is supposed the Hunter is always initiated in the
        corner and oriented
        ls.setFloodlight(true);
        actuator.rotate(90);
        ls.readValue();
        ls.calibrateLow();
        ls.setLow(0);
        actuator.rotate(90);
        ls.readValue();
        ls.calibrateHigh();
        ls.setHigh(100);
        actuator.rotate(180);
        Hunter.Zigzag();
        if (ls.readValue() > 50)
        {
            Hunter.Hunting();
        }
    }
}

```

LINEFOLLOWER 1

Daniel Egido Sánchez de Vega



- Follower

+++++

```
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.SensorConstants;

public class Follower
{
    public static void main (String[] args)
    {
        SensorPort.S1.setTypeAndMode(SensorConstants.TYPE_LIGHT_
ACTIVE,
        SensorConstants.MODE_PCTFULLSCALE);
        while(true)
        {
            if(SensorPort.S1.readValue() > 50)
            {
                Motor.A.setSpeed(75);
                Motor.C.setSpeed(0);
            }else{
                Motor.A.setSpeed(0);
                Motor.C.setSpeed(75);
            }
        }
    }
}
```

LINEFOLLOWER 2

Daniel Egido Sánchez de Vega



- Follower

+++++

```
import lejos.nxt.*;
```

```
public class Follower {
```

```
    public static void main (String[] aArg)
    throws Exception
    {
```

```
        String left = "Left ";
        String right= "Right";
```

```
        LightSensor light = new LightSensor(SensorPort.S3);
        final int blackWhiteThreshold = 45;
```

```
        final int forward = 1;
        final int stop = 3;
        final int flt = 4;
        final int power = 80;
```

```
        // Use the light sensor as a reflection sensor
        light.setFloodlight(true);
        LCD.drawString("Light %: ", 0, 0);
```

```
        // Show light percent until LEFT is pressed
        while (! Button.LEFT.isPressed()){
            LCD.drawInt(light.readValue(), 3, 9, 0);
            LCD.refresh();
        }
```

```
        // Follow <b style="color:black;background-
color:#ffff66">line</b> until ESCAPE is pressed
```

```
        while (! Button.ESCAPE.isPressed()){
```

```
            if (light.readValue() > blackWhiteThreshold){
                // On white, turn right
                LCD.drawString(right, 0, 1);
                MotorPort.B.controlMotor(0,stop);
                MotorPort.C.controlMotor(power,
```

```
forward);
```

```
            }
```

```
            else{
```

```
                // On black, turn left
                LCD.drawString(left, 0, 1);
```



```
        MotorPort.B.controlMotor(power,
forward);
        MotorPort.C.controlMotor(0,stop);
    }
    LCD.drawInt(light.readValue(), 3, 9, 0);
    LCD.refresh();
    Thread.sleep(100);

}

// Stop car gently with free wheel drive
MotorPort.B.controlMotor(0,flt);
MotorPort.C.controlMotor(0,flt);
LCD.clear();
LCD.drawString("Program stopped", 0, 0);
LCD.refresh();
Thread.sleep(1000);
}
}
```

TREMAUX MAZE SOLVER

Daniel Egido Sánchez de Vega



- Sensing (Idem)
- Actuator (Idem)
- Solver

+++++

```

import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class Solver
{
    static UltrasonicSensor us1 = new
    UltrasonicSensor(SensorPort.S1);
    static UltrasonicSensor us2 = new
    UltrasonicSensor(SensorPort.S2);
    static UltrasonicSensor us3 = new
    UltrasonicSensor(SensorPort.S3);
    static Actuator actuator = new Actuator();
    static Sensing sensing = new Sensing();

    //Direction Mydir = new Direction();
    static int MyDir;
    //define the directions in space
    static int North = 0;
    static int West = 1;
    static int South = 2;
    static int East = 3;

    static int LeftOf;
    static int RightOf;

    //relative positions
    static int RelPosX;
    static int RelPosY;

    //Size of maze map
    static int SizeX = 10;
    static int SizeY = 10;
    static int Map[][] = new int [SizeX][SizeY];
    // 0      position which is not visited or cannot be visited
    // 1      position whose all ways are tried
    // 2      position already visited but one way not tried
    // 3      position already visited but two ways not tried

    //hold path traveled
    static int MyPath[] = new int [SizeX*SizeY];

```



```

//start position
static int MyPosX = 6;
static int MyPosY = 0;

static int PathCounter = 0;

static void Left_of (int dir)
{
    LeftOf = (dir+1) % 4;
}

static void Right_of (int dir)
{
    if(dir == North)
    {
        RightOf = East;
    }else{
        RightOf-= 1;
    }
}

//Convert directions to relative positions in map
static void RelX(int dir)
{
    if(dir == West)
    {
        RelPosX = -1;
    }else if(dir == East){
        RelPosX = 1;
    }else{
        RelPosX = 0;
    }
}

static void RelY(int dir)
{
    if(dir == North)
    {
        RelPosY = 1;
    }else if(dir == South){
        RelPosY = -1;
    }else{
        RelPosY = 0;
    }
}

//Current Position
//Map[MyPosX][MyPosY]
//currently in front of us

```



```

//Map[MyPosX + RelPosX(MyDir)][MyPosY + RelPosY(MyDir)]
//current right
//Map(MyPosX + RelPosX(RightOf(MyDir)))[MyPosY +
RelPosY(RightOf(MyDir))]
//current left
//Map(MyPosX + RelPosX(LeftOf(MyDir)))[MyPosY +
RelPosY(LeftOf(MyDir))]

//Initialize map
static void InitialMap()
{
    //boundaries and inside maze
    int i = 0;
    int j = 0;
    for (i=0;i<SizeX;i++)
    {
        Map[i][0] = Map[i][SizeY-1] = 1;
    }
    for (j=0;j<SizeY;j++)
    {
        Map[0][j] = Map[SizeX-1][j] = 1;
    }
    for (i=1;i<SizeX-1;i++)
    {
        for (j=1;j<SizeY-1;j++)
        {
            Map[i][j] = Map[i][j] = 0;
        }
    }
}

static boolean Exit()
{
    if (MyPosX == 6 && MyPosY == 9)
    {
        return true;
    }else{
        return false;
    }
}

//Decreasing one
static void Dec_One(int var)
{
    var -=1;
    if(var < 1)
    {
        var = 1;
    }
}

```



```

//checking where is the near wall
static int Wall[] = new int[3];
// 0    left;
// 1    front
// 2    right
static int NextMove;           //which is the suitable next
move
static int NearWall = 20; //distance to near wall

public static void main (String[] args)
{
    while(!Exit())           //while robot doesn't reach the exit
position
    {
        Wall[0]=Wall[1]=Wall[2]=0;
        //check which direction is the wall near
        //check left
        if(sensing.distance(us2) <= NearWall)
        {
            Wall[0] = 1;
        }
        //check front
        if(sensing.distance(us1) <= NearWall)
        {
            Wall[1] = 1;
        }
        //check right
        if(sensing.distance(us3) <= NearWall)
        {
            Wall[2] = 1;
        }

        //firstly assuming all ways are unvisited
        Map[MyPosX][MyPosY] = 3 - Wall[0] - Wall[1] -
Wall[2];
        NextMove = -1; //all ways are visited

        Left_of(MyDir);
        Right_of(MyDir);
        RelX(MyDir);
        RelY(MyDir);

        //check left path
        if(Wall[0] == 0)
        {
            RelX(LeftOf);
            RelY(LeftOf);
            //have we visited the left position?

```



```

        if(Map[MyPosX + RelPosX][MyPosY +
RelPosY] == 0)
        {
            NextMove = 0;
        }else{
            Dec_One(Map[MyPosX][MyPosY]);
            Dec_One(Map[MyPosX +
RelPosX][MyPosY + RelPosY]);
        }
    }

    //check front path
    if(Wall[1] == 0)
    {
        RelX(MyDir);
        RelY(MyDir);
        //have we visited the left position?
        if(Map[MyPosX + RelPosX][MyPosY +
RelPosY] == 0)
        {
            NextMove = 1;
        }else{
            Dec_One(Map[MyPosX][MyPosY]);
            Dec_One(Map[MyPosX +
RelPosX][MyPosY + RelPosY]);
        }
    }

    //check right path
    if(Wall[2] == 0)
    {
        RelX(RightOf);
        RelY(RightOf);
        //have we visited the left position?
        if(Map[MyPosX + RelPosX][MyPosY +
RelPosY] == 0)
        {
            NextMove = 2;
        }else{
            Dec_One(Map[MyPosX][MyPosY]);
            Dec_One(Map[MyPosX +
RelPosX][MyPosY + RelPosY]);
        }
    }

    switch(NextMove)
    {
        case 0:                // go left
                                actuator.Left();
                                Left_of(MyDir);

```



```

                                MyDir = LeftOf;
                                break;
        case 2:                // go right
                                actuator.Right();
                                Right_of(MyDir);
                                MyDir = RightOf;
                                break;
    }

    if(NextMove != -1)          //Should we move forward?
    {
        actuator.Travel();
        MyPath[PathCounter++] = NextMove;
        RelX(MyDir);
        MyPosX = RelPosX;
        RelY(MyDir);
        MyPosY = RelPosY;
        continue;
    }

    actuator.back();

    while(Map[MyPosX][MyPosY] < 2 && PathCounter-- >
0)
    {
        switch(MyPath[PathCounter])
        {
            case 0:            //go back right
                                actuator.Right();
                                Right_of(MyDir);
                                MyDir = RightOf;
                                break;

            case 2:            //go back left
                                actuator.Left();
                                Left_of(MyDir);
                                MyDir = LeftOf;
                                break;

        }
        actuator.Travel();
        RelX(MyDir);
        RelY(MyDir);
        //change position
        MyPosX += RelPosX;
        MyPosY += RelPosY;
    }
    }
    actuator.stop();
}
}
}

```



INTERFACE:**RPS NXT:**

- **RPSSystem**
- **Constants**
 - **Event**
 - **Protocol**
 - **State**
- **Localizer**
 - Encoder**
 - Localizer**
- **Data**
 - **EncoderData**
 - **Range**
 - **SonarData**
 - **TrueFalse**
- **Utilities**
 - **Current Position**
 - **Direction Calculator**
 - **Detection**
 - **Range**
 - **Timer**
 - **Control**
 - **Coordinator**
 - **Sonar Sensor**
 - **Speaker**
 - **Actuator**
- **RPS Server**
- **RPS Server Listener**

+++++



```

import Localizer.Localizer;
import rpss_utilities.CurrPosition;
import rpss_utilities.RPCoordinator;
import rpss_data.EncoderData;
import selab.rpss.RobotPatrolSystemServer;
import selab.rpss.RobotPatrolSystemServerListener;

public class RPSSystem implements RobotPatrolSystemServerListener
{

    private RobotPatrolSystemServer server = null;
    private RPCoordinator RPC;
    private CurrPosition CP;
    private EncoderData ED;
    private Localizer LT;

    public RPSSystem()
    {
        server = new RobotPatrolSystemServer();
        server.setListener(this);
        CP = new CurrPosition();
        ED = new EncoderData();
        LT = new Localizer(CP, server, ED);
        RPC = new RPCoordinator(LT, server, CP);

    }

    public void start()
    {
        server.startServer(RPC);
        RPC.timer.start();
    }

    public void startPatrol(int x, int y)
    {
        server.debug("start Patrol x = "+x+", y = "+y);
        RPC.startPatrol(x, y);
    }

    public void stopPatrol()
    {
        server.debug("stopPatrol called!!");
        RPC.stopPatrol();
    }
}

```



```

    public static void main(String args[])
    {
        RPSSystem rpsSystem= new RPSSystem();

        rpsSystem.start();
    }
}

+++++

package Constants;

public class Event {
    public static final int EMPTY = 0;
    public static final int START_PATROL = 1;

    public static final int SETTING = 2;

    public static final int GET_DIRECTION = 70;

    public static final int DIRECTION = 5;
    public static final int TURN_DIRECTION = 6;
    public static final int TURNED = 7;
    public static final int START = 8;
    public static final int STARTED = 9;

    public static final int STARTING = 10;

    public static final int AFTER_ELAPEDTIME = 14;
    public static final int READ_SENSORDATA = 15;
    public static final int SENSORDATA = 16;
    public static final int CHECK_INTRUSION = 17;
    public static final int NO_INTRUSION = 18;
    public static final int GET_CURRPOSITION = 19;
    public static final int CURRPOSITOIN = 20;
    public static final int IS_ARRIVED = 21;
    public static final int NO_DESTINATION = 22;
    public static final int YES_DESTINATION = 23;
    public static final int YES_INTRUSTION = 24;

    public static final int STOP_1 = 25;
    public static final int STOP_2 = 26;
    public static final int STOP_3 = 27;

    public static final int
    STOPPED_NOINTRUTION_YESDESTINATION = 28;

```



```

        public static final int UPDATE_RANGE = 29;
        public static final int UPDATED_RANGE = 36;
        public static final int STOPPED_YESINTRUSION = 30;
        public static final int ALARM = 31;
        public static final int ALARMED = 32;
        public static final int PATROL_STOPPED = 33;
        public static final int STOPPED_NOINTRUSION_NODEPARTED
= 34;
        public static final int STOP_PATROL = 35;

        public static final int AVOID_TIMER = 100;

        public static final int NO_defined = 444;
    }

+++++

package Constants;

/**
 * RobotPatroSubSystem
 *
 */
public class Protocol
{
    public static final int PROTOCOL_DUMMY = 999;
    public static final int PROTOCOL_CLOSECONNECTION= 000;
    public static final int PROTOCOL_STARTPATROL = 100;
    public static final int PROTOCOL_STOPPATROL = 200;
    public static final int PROTOCOL_PATROLSTOPPED = 300;
    public static final int PROTOCOL_DEBUG = 400;
    public static final int PROTOCOL_RESULTOK = 800;
    public static final int PROTOCOL_RESULTERROR = 900;
    public static final int PROTOCOL_ISIDLE = 777;
}

+++++

package Constants;

/**
 *
 * <<state dependent control>> RobotPatrolControl
 *
 */
public class State {
    public static final int IDLE = 900;
    public static final int CALCULATING_DIRECTION = 901;
    public static final int TURNING_DIRECTION = 902;

```



```
public static final int STARTING = 903;  
public static final int MOVING = 904;  
public static final int READING_SENSOR = 905;  
public static final int CHECKING_INTRUSION = 906;  
public static final int LOCALIZING = 907;  
public static final int CHECKING_DESTINATION = 908;  
public static final int STOPPING = 909;  
public static final int ALARMING = 912;  
public static final int UPDATING_RANGE = 913;  
}
```



```

package Localizer;

import lejos.navigation.Pilot;
import lejos.nxt.Motor;
import rpss_data.EncoderData;

public class Encoder {
    private Pilot pilot = new Pilot(5.6f, 11.3f, Motor.C, Motor.A, false);
    public EncoderData ED;
    public Encoder(EncoderData ED_){
        ED = ED_;
    }

    public void readEncoder()
    {
        ED.rightcount = pilot.getRightCount();
        ED.leftcount = pilot.getLeftCount();
        ED.distance = pilot.getTravelDistance();
    }
}

+++++
package Localizer;

import rpss_data.EncoderData;
import rpss_utilities.CurrPosition;
import selab.rpss.RobotPatrolSystemServer;
import lejos.util.Timer;
import lejos.util.TimerListener;

public class Localizer implements TimerListener{

    public Timer timer = new Timer(50, this);
    private Encoder EN;
    private EncoderData ED;
    private CurrPosition CP;

    public Localizer(CurrPosition CP_, RobotPatrolSystemServer USR,
EncoderData ED_){
        CP = CP_;

        ED = ED_;
        EN = new Encoder(ED);
    }
    public void timedOut() {
        localize();
    }
}

```



```
    public void localize()
    {
        EN.readEncoder();
        CP.setCurrPosition((int)ED.distance);
    }
}
```



```

package rpss_data;

public class EncoderData {
    public int rightcount;
    public int leftcount;
    public float distance;
    public EncoderData(){

    }
}

+++++
package rpss_data;

public class Range {
    public int start_x;
    public int start_y;
    public int end_x;
    public int end_y;
}

+++++
package rpss_data;

public class SonarData {

    private int distance;
    public SonarData(){
        distance = 10000;
    }

    public void setDistance(int d){
        distance = d;
    }
    public int getDistance(){
        return distance;
    }
}

+++++
package rpss_data;

public class TrueFalse {
    public boolean flag;
}

```



```
package rpss_utilities;
```

```
public class CurrPosition {

    private int distance;
    public CurrPosition(){
        distance = 0;
    }

    public void setCurrPosition(int dis_){
        distance = dis_;
    }
    public int getCurrPosition(){
        return distance;
    }
}
```

```
+++++
```

```
package rpss_utilities;
```

```
public class DirectionCalculator
{
    public DirectionCalculator()
    {

    }
    public int getDirection(int start_x, int start_y, int end_x, int end_y)
    {
        return ((int)Math.toDegrees(Math.atan2((end_y - start_y),
(end_x - start_x)))-90);
    }
}
```

```
+++++
```

```
package rpss_utilities;
```

```
public class IntrusionDetector {

    public IntrusionDetector(){

    }
    public boolean checkIntrusion(int distance){
        if ( distance < 15 ) return true;
        else return false;
    }
}
```



```

}

+++++
package rpss_utilities;

import rpss_data.Range;

public class PatrolRange {

    private int distance;
    private Range R;
    public PatrolRange(Range R_){
        R = R_;
    }
    public void setRange(int x, int y){
        R.start_x = 0;
        R.start_y = 0;
        R.end_x = x;
        R.end_y = y;

        distance = (int) Math.sqrt((R.start_x - R.end_x) * (R.start_x -
R.end_x) + (R.start_y - R.end_y) * (R.start_y - R.end_y));
    }
    public boolean isArrived(int distance_){
        if ( distance_ > distance )return true;
        else return false;
    }
    public int thisdistance(){
        return distance;
    }
    public void updateRange(){
        int temp;
        temp = R.start_x;
        R.start_x = R.end_x;
        R.end_x = temp;

        temp = R.start_y;
        R.start_y = R.end_y;
        R.end_y = temp;
    }
}

+++++
package rpss_utilities;

import rpss_data.TrueFalse;
import lejos.util.Timer;
import lejos.util.TimerListener;

```



```

/**
 *
 * Every 50ms, PatrolTimer class set the variable flag of TF to true
 * Coordinator, maybe, references this flag.
 */
public class PatrolTimer implements TimerListener{

    public Timer timer = new Timer(100, this);
    public TrueFalse timerFlag;

    public PatrolTimer(TrueFalse flag){
        timerFlag = flag;
    }

    public void timedOut() {
        timerFlag.flag = true;
    }
}

+++++
package rpss_utilities;

import Constants.Event;
import Constants.State;

public class RobotPatrolControl {

    public int state = State.IDLE;

    public RobotPatrolControl(){

    }

    public int processEvent(int event){

        if ( event == Event.AFTER_ELAPEDTIME && state !=
State.MOVING )
            return Event.AVOID_TIMER;
        else{
            if (state == State.IDLE ){

                if ( event == Event.START_PATROL ){
                    state =
State.CALCULATING_DIRECTION;
                    return Event.SETTING;
                }
            }
        }
    }
}

```



```

else if (state == State.CALCULATING_DIRECTION ){

    if ( event == Event.DIRECTION ) {
        state = State.TURNING_DIRECTION;
        return Event.TURN_DIRECTION;
    }
}
else if (state == State.TURNING_DIRECTION ){

    if ( event == Event.TURNED ) {
        state = State.STARTING;
        return Event.START;
    }
}
else if (state == State.STARTING ){

    if ( event == Event.STARTED ){
        state = State.MOVING;
        return Event.STARTING;
    }
}
else if (state == State.MOVING ){

    if ( event == Event.AFTER_ELAPEDTIME ){
        state = State.READING_SENSOR;
        return Event.READ_SENSORDATA;
    }
    else if ( event == Event.STOP_PATROL){
        state = State.STOPPING;
        return Event.STOP_2;
    }
}
else if (state == State.READING_SENSOR ){

    if ( event == Event.SENSORDATA ){
        state = State.CHECKING_INTRUSION;
        return Event.CHECK_INTRUSION;
    }
    else if ( event == Event.STOP_PATROL){
        state = State.STOPPING;
        return Event.STOP_2;
    }
}
else if (state == State.CHECKING_INTRUSION ){

    if ( event == Event.NO_INTRUSION ){
        state = State.LOCALIZING;
        return Event.GET_CURRPOSITION;
    }
}

```



```

        else if ( event == Event.STOP_PATROL){
            state = State.STOPPING;
            return Event.STOP_2;
        }
        else if ( event == Event.YES_INTRUSTION){
            state = State.STOPPING;
            return Event.STOP_1;
        }
    }
    else if (state == State.LOCALIZING ){

        if ( event == Event.CURRPOSITOIN ){
            state =
State.CHECKING_DESTINATION;
            return Event.IS_ARRIVED;
        }
        else if ( event == Event.STOP_PATROL){
            state = State.STOPPING;
            return Event.STOP_2;
        }
    }
    else if (state == State.CHECKING_DESTINATION ){

        if ( event == Event.NO_DESTINATION ){
            state = State.MOVING;
            return Event.EMPTY;
        }
        else if ( event == Event.YES_DESTINATION){
            state = State.STOPPING;
            return Event.STOP_3;
        }
        else if ( event == Event.STOP_PATROL){
            state = State.STOPPING;
            return Event.STOP_2;
        }
    }
    else if (state == State.STOPPING ){

        if ( event == Event.STOPPED_YESINTRUSION
){
            state = State.ALARMING;
            return Event.ALARM;
        }
        else if ( event ==
Event.STOPPED_NOINTRUSION_NODEPARTED){
            state = State.IDLE;
            return Event.PATROL_STOPPED;
        }
        else if ( event ==
Event.STOPPED_NOINTRUTION_YESDESTINATION){

```



```

        state = State.UPDATING_RANGE;
        return Event.UPDATE_RANGE;
    }
}
else if (state == State.ALARMING ){
    if ( event == Event.ALARMED ){
        state = State.IDLE;
        return Event.PATROL_STOPPED;
    }
}
else if (state == State.UPDATING_RANGE ){
    if ( event == Event.UPDATED_RANGE ){
        state =
State.CALCULATING_DIRECTION;
        return Event.GET_DIRECTION;
    }
}
}
return event;
}
}

```

```

+++++
package rpss_utilities;

```

```

import Constants.Event;
import Constants.State;
import Localizer.Localizer;
import rpss_data.Range;
import rpss_data.SonarData;
import rpss_data.TrueFalse;
import selab.rpss.RobotPatrolSystemServer;
import lejos.util.Timer;
import lejos.util.TimerListener;

```

```

/**
 *
 * RPCoordinator coordinate events and actions
 * using variable action returned from RobotPatrolControl
 * that has parameter : event
 */
public class RPCoordinator implements TimerListener{

    private RobotPatrolSystemServer server;

    public Timer timer = new Timer(20, this);

```



```

        public TrueFalse timerElapsedflag ; //For recognizing PatrolTimer
        fired.

```

```

        private Object syncObj = new Object(); //Monitor used for
        coordinate().

```

```

        private int x, y;
        private int action, temp_action;

```

```

        private boolean stopFlag = false;

```

```

        private int direction;
        //private int currentpositionX, currentpositionY;
        private int currentDistance;
        private DirectionCalculator DC;
        private IntrusionDetector ID;
        private PatrolRange PR;
        private PatrolTimer PT;
        private Localizer LT;
        private RobotPatrolControl RPC;
        private SonarSensor SS;
        private Speaker SPK;
        private WheelActuator WA;
        private CurrPosition CP;
        private Range R;

```

```

        private SonarData SD;

```

```

        public RPCoordinator(Localizer LT_, RobotPatrolSystemServer
        USR, CurrPosition CP_){

```

```

            server = USR; // For debug.
            CP = CP_;

```

```

            timerElapsedflag = new TrueFalse();
            timerElapsedflag.flag = false;
            action = Event.EMPTY;

```

```

            LT = LT_;
            DC = new DirectionCalculator();
            ID = new IntrusionDetector();
            R = new Range();
            PR = new PatrolRange(R);
            PT = new PatrolTimer(timerElapsedflag);

```

```

            RPC = new RobotPatrolControl();

```



```

        SD = new SonarData();
        SS = new SonarSensor(SD);
        SPK = new Speaker();
        WA = new WheelActuator();
    }

private void coordinate(){
    if ( RPC.state == State.IDLE)    server.isIdle(true);
    else server.isIdle(false);

    if ( timerElapsedflag.flag && RPC.state == State.MOVING ){
        timerElapsedflag.flag = false;
        action = Event.AFTER_ELAPEDTIME;
    }
    if (    stopFlag &&
        (
            RPC.state == State.MOVING ||
            RPC.state == State.READING_SENSOR
||
            RPC.state ==
State.CHECKING_INTRUSION ||
            RPC.state ==
State.CHECKING_DESTINATION ||
            RPC.state == State.LOCALIZING
        )
    )
    {
        stopFlag = false;
        action = Event.STOP_PATROL;
    }

    if ( action == Event.AVOID_TIMER ) {
        action = temp_action;
        return;
    }

    if ( action == Event.SETTING ){
        action = Event.DIRECTION;

        PR.setRange(x, y);
        CP.setCurrPosition(0);
        direction = DC.getDirection(0, 0, x, y);
        server.debug("x : "+x+" y : "+y);
        server.debug("Direction : "+direction);
    }
    else if ( action == Event.TURN_DIRECTION ){

```



```

        action = Event.TURNED;
        WA.Rotate(direction);

    }
    else if ( action == Event.START ){
        action = Event.STARTED;
        WA.Forward();

    }

    else if ( action == Event.STARTING ){
        action = Event.EMPTY;
        PT.timer.start();
        LT.timer.start();
        // 'Departed message' is needed.

    }

    else if ( action == Event.READ_SENSORDATA ){
        action = Event.SENSORDATA;

        SS.readSonarData();
    }
    else if ( action == Event.CHECK_INTRUSION ){
        if ( ID.checkIntrusion(SD.getDistance()) ) action =
Event.YES_INTRUSTION;
        else action = Event.NO_INTRUSION;
    }
    else if ( action == Event.GET_CURRPOSITION ){
        action = Event.CURRPOSITOIN;
        // currentpositionX = CP.getCurrPositionX();
        // currentpositionY = CP.getCurrPositionY();
        currentDistance = CP.getCurrPosition();
        //server.debug("(cur_dis)[["+currentDistance +
""]](abs_dis)[["+PR.thisdistance()+"]]");
    }
    else if ( action == Event.IS_ARRIVED ){
        if ( PR.isArrived(currentDistance)) action =
Event.YES_DESTINATION;
        else action = Event.NO_DESTINATION;

    }
    else if ( action == Event.STOP_1 ){

        action = Event.STOPPED_YESINTRUSION;
        PT.timer.stop();
        LT.timer.stop();
        WA.Stop();

    }

```



```

        else if ( action == Event.STOP_2 ){
            action =
Event.STOPPED_NOINTRUSION_NODEPARTED;
            PT.timer.stop();
            LT.timer.stop();
            WA.Stop();
        }
        else if ( action == Event.STOP_3 ){
            action =
Event.STOPPED_NOINTRUSION_YESDESTINATION;
            PT.timer.stop();
            LT.timer.stop();
            WA.Stop();
        }
        else if ( action == Event.UPDATE_RANGE ){
            action = Event.UPDATED_RANGE;
            PR.updateRange();
        }
        else if ( action == Event.ALARM ){
            action = Event.ALARMED;
            SPK.Beep();
        }
        else if ( action == Event.PATROL_STOPPED ){
            action = Event.EMPTY;
        }
        else if ( action == Event.GET_DIRECTION){
            action = Event.DIRECTION;
            //direction = DC.getDirection(R.start_x, R.start_y,
R.end_x, R.end_y);
            direction = 190;
        }
    }

    public void startPatrol(int p, int q){
        x = p;
        y = q;
        stopFlag = false;
        action = Event.START_PATROL;
    }

    public void stopPatrol(){
        stopFlag = true;
    }
}

```



```

        public void timedOut() {
            synchronized(syncObj){
                coordinate();
            }
        }
    }

+++++
package rpss_utilities;

import rpss_data.SonarData;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class SonarSensor {
    private UltrasonicSensor sonic = new
UltrasonicSensor(SensorPort.S1);
    private SonarData sonar;

    public SonarSensor(SonarData SD){
        sonar = SD;
    }
    public void readSonarData(){
        sonar.setDistance(sonic.getDistance());
    }
}

+++++
package rpss_utilities;

import lejos.nxt.Sound;

public class Speaker {
    public void Beep(){
        Sound.beep();
    }
}

+++++
package rpss_utilities;

import lejos.navigation.Pilot;
import lejos.nxt.Motor;

public class WheelActuator {

```



```
private Pilot pilot = new Pilot(5.6f, 11.3f, Motor.C, Motor.A, false);

public void Forward(){
    pilot.forward();
}
public void Stop(){
    pilot.stop();
    pilot.resetTachoCount();
}
public void Rotate(int angle){
    pilot.rotate(angle);
}
}
```



```

package selab.rpss;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;

import Constants.Protocol;

import rpss_utilities.RPCoordinator;

import lejos.nxt.LCD;
import lejos.nxt.comm.BTConnection;
import lejos.nxt.comm.Bluetooth;

/**
 *
 *
 * Bluetooth
 */
public class RobotPatrolSystemServer
{
    private Object syncObj = new Object();

    private RobotPatrolSystemServerListener listener = null;
    private BTConnection btc;
    private DataInputStream dis;
    private DataOutputStream dos;

    public RobotPatrolSystemServerListener getListener()
    {
        return listener;
    }

    /**
     *
     *
     * param listener
     */
    public void setListener(RobotPatrolSystemServerListener listener)
    {
        this.listener = listener;
    }

    /**
     *
     * Bluetooth Connection
     */
    public void startServer(RPCoordinator RPC)

```



```

{
    while(true)
    {
        LCD.clear();
        LCD.drawString("Server Started!!", 0, 0);

        btc = Bluetooth.waitForConnection();

        LCD.clear();
        LCD.drawString("Client Connected", 0, 0);

        try
        {
            dis = btc.openDataInputStream();
            dos = btc.openDataOutputStream();

            RPC.timer.start();

            while (true)
            {
                readDummy();
                int command = dis.readInt();

                if (command ==
Protocol.PROTOCOL_CLOSECONNECTION)
                {
                    btc.closeStream();
                    btc.close();
                    break;
                }
                if (command ==
Protocol.PROTOCOL_STARTPATROL)
                {
                    int x = dis.readInt();
                    int y = dis.readInt();
                    if(listener != null)
                    {
                        //test
                        listener.startPatrol(x, y);
                        synchronized(syncObj)
                        {
                            writeDummy();

                            dos.writeInt(Protocol.PROTOCOL_RESULTOK);
                            dos.flush();
                        }
                        LCD.clear();
                        LCD.drawString("Start
Patrol", 0, 0);
                    }
                }
            }
        }
    }
}

```



```

        else
        {
            synchronized(syncObj)
            {
                writeDummy();

                dos.writeInt(Protocol.PROTOCOL_RESULTERROR);
                dos.flush();
            }
        }
    }
    else if (command ==
Protocol.PROTOCOL_STOPPATROL)
    {
        if(listener != null)
        {
            listener.stopPatrol();
            synchronized(syncObj)
            {
                writeDummy();

                dos.writeInt(Protocol.PROTOCOL_RESULTOK);
                dos.flush();
            }
        }
        else
        {
            synchronized(syncObj)
            {
                writeDummy();

                dos.writeInt(Protocol.PROTOCOL_RESULTERROR);
                dos.flush();
            }
        }
    }
}
catch (IOException e)
{
    LCD.clear();
    LCD.drawString(e.getMessage(), 0, 0);
    btc.closeStream();
    btc.close();
}
}
}

```



```

private void readDummy() throws IOException
{
    dis.readInt();
    dis.readInt();
    dis.readInt();
}

private void writeDummy() throws IOException
{
    synchronized(syncObj)
    {
        dos.writeInt(Protocol.PROTOCOL_DUMMY);
        dos.writeInt(Protocol.PROTOCOL_DUMMY);
        dos.writeInt(Protocol.PROTOCOL_DUMMY);
    }
}

public void patrolStopped()
{
    try
    {
        synchronized(syncObj)
        {
            writeDummy();

            dos.writeInt(Protocol.PROTOCOL_PATROLSTOPPED);
            dos.flush();
        }
    }
    catch(Exception e)
    {
        LCD.clear();
        LCD.drawString("1"+e.getMessage(), 0, 0);
    }
}

/**
 * String
 * param msg
 */
public void debug(String msg)
{
    try
    {
        synchronized(syncObj)
        {
            writeDummy();
            dos.writeInt(Protocol.PROTOCOL_DEBUG);
            dos.writeInt(msg.length());

```



```

        dos.writeChars(msg);
        dos.flush();
    }
}
catch(Exception e)
{
    LCD.clear();
    LCD.drawString("1"+e.getMessage(), 0, 0);
}
}

public void isIdle(boolean idleFlag)
{
    try
    {
        synchronized(syncObj)
        {
            writeDummy();
            dos.writeInt(Protocol.PROTOCOL_ISIDLE);
            dos.writeBoolean(idleFlag);
            dos.flush();
        }
    }
    catch(Exception e)
    {
        LCD.clear();
        LCD.drawString("1"+e.getMessage(), 0, 0);
    }
}
}

```

```

+++++

```

```

package selab.rpss;

```

```

/**

```

```

 * Patrol Commander

```

```

 *

```

```

 */

```

```

public interface RobotPatrolSystemServerListener

```

```

{

```

```

    /**

```

```

    *param x

```

```

    *param y

```

```

    */

```

```

    void startPatrol(int x, int y);

```

```

    void stopPatrol();

```

```

}

```



INTERFACE**RPX GUI**

- **UserInterfaceSubsystem**
- **Protocol**
- **RPS Agent**
- **RPS Listener**

+++++

```
package gui;
```

```
import guiutil.rpsa.RobotPatrolSystemAgent;
import guiutil.rpsa.RobotPatrolSystemListener;
```

```
import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
```

```
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRootPane;
import javax.swing.JTextField;
```

```
public class UserInterfaceSubSystem extends JFrame implements
WindowListener,
        ActionListener, RobotPatrolSystemListener
{
```

```
    private static final long serialVersionUID = 1L;
    private RobotPatrolSystemAgent agent;
    private JButton btnStart;
    private JButton btnStop;
    private JButton btnConnect;
    private JButton btnDisconnect;
    private JTextField cName;
    private JTextField cAddress;
    private JTextField xDest;
    private JTextField yDest;
```

Daniel Egido Sánchez de Vega



```

private boolean idleFlag_ = true;

public UserInterfaceSubSystem()
{
    agent = new RobotPatrolSystemAgent();
    agent.setListener(this);
    initialize();
}

/**
 * UI Control
 */
private void initialize()
{
    JRootPane contentPane = this.getRootPane();
    BorderLayout layoutManager = new BorderLayout(10, 10);

    contentPane.setLayout(layoutManager);

    contentPane.add(createInputPane(), BorderLayout.NORTH);
    contentPane.add(createButtonPane(),
BorderLayout.SOUTH);

    this.setSize(400, 230);
    this.setLocation(300, 300);
}

private Component createInputPane()
{
    JPanel panel = new JPanel();
    panel.setLayout(new BorderLayout(5, 5));
    panel.add(createNXTNameInputPane(),
BorderLayout.NORTH);
    panel.add(createNXTDestinationInputPane(),
BorderLayout.CENTER);
    panel.add(createNXTAddressInputPane(),
BorderLayout.SOUTH);
    return panel;
}

private Component createNXTDestinationInputPane(){
    JPanel panel = new JPanel();
    xDest = new JTextField("0", 5);
    yDest = new JTextField("10", 5);

    panel.setLayout(new FlowLayout());
    panel.add(new JLabel("X : "));
    panel.add(xDest);
    panel.add(new JLabel("Y : "));

```



```

        panel.add(yDest);

        return panel;
    }

    private Component createNXTNameInputPane()
    {
        JPanel panel = new JPanel();
        cName = new JTextField("DANIMASTER", 5);

        panel.setLayout(new FlowLayout());
        panel.add(new JLabel("NXT ConInfo : "));
        panel.add(new JLabel(" Name : "));
        panel.add(cName);

        return panel;
    }

    private Component createNXTAddressInputPane()
    {
        JPanel panel = new JPanel();
        //cAddress = new JTextField("001653039bb0", 12);
        cAddress = new JTextField("00165302809a", 12);

        panel.setLayout(new FlowLayout());
        panel.add(new JLabel("Address : "));
        panel.add(cAddress);

        return panel;
    }

    private Component createButtonPane()
    {
        JPanel panel = new JPanel();
        BoxLayout layoutManager = new BoxLayout(panel,
BoxLayout.X_AXIS);
        btnConnect = new JButton();
        btnDisconnect = new JButton();
        btnStart = new JButton();
        btnStop = new JButton();

        panel.setLayout(layoutManager);
        panel.add(Box.createHorizontalGlue());
        panel.add(btnConnect);
        panel.add(btnDisconnect);
        panel.add(btnStart);
        panel.add(btnStop);

        btnConnect.addActionListener(this);

```



```

        btnDisconnect.addActionListener(this);
        btnConnect.setText("Connection");
        btnDisconnect.setText("Disconnection");
        btnDisconnect.setEnabled(false);

        btnStart.addActionListener(this);
        btnStop.addActionListener(this);
        btnStart.setText("Patrol");
        btnStop.setText("STOP");
        btnStart.setEnabled(false);
        btnStop.setEnabled(false);
        return panel;
    }

    /**
     * Patrol
     */
    private void startPatrol()
    {
        agent.startPatrol( Integer.parseInt(xDest.getText()),
Integer.parseInt(yDest.getText()) );
        btnStart.setEnabled(false);
        btnStop.setEnabled(true);
    }

    /**
     * Patrol
     */
    public void stopPatrol()
    {
        agent.stopPatrol();

        btnStop.setEnabled(false);
        btnStart.setEnabled(true);
    }

    public void Connect(){
        String connectName = cName.getText();
        String connectAddress = cAddress.getText();
        agent.connect(connectName, connectAddress);

        if ( idleFlag_ ){
            btnStart.setEnabled(true);
            btnStop.setEnabled(false);
        }
    }

```



```

        else {
            btnStart.setEnabled(false);
            btnStop.setEnabled(true);
        }

        btnConnect.setEnabled(false);
        btnDisconnect.setEnabled(true);
    }
    public void Disconnect(){
        agent.close();
        btnStart.setEnabled(false);
        btnStop.setEnabled(false);

        btnConnect.setEnabled(true);
        btnDisconnect.setEnabled(false);
    }

    // WindowListener ±,Çö 0°Ð
    // -----
    public void windowClosing(WindowEvent e)
    {
        dispose();
        System.exit(0);
    }

    public void windowOpened(WindowEvent e)
    {
    }

    public void windowActivated(WindowEvent e)
    {
    }

    public void windowIconified(WindowEvent e)
    {
    }

    public void windowDeiconified(WindowEvent e)
    {
    }

    public void windowDeactivated(WindowEvent e)
    {
    }

    public void windowClosed(WindowEvent e)
    {
    }

    public void actionPerformed(ActionEvent event)

```



```

    {
        if (event.getSource().equals(btnStart))
        {
            startPatrol();
        }
        else if (event.getSource().equals(btnStop))
        {
            stopPatrol();
        }
        else if (event.getSource().equals(btnConnect))
        {
            Connect();
        }
        else if (event.getSource().equals(btnDisconnect))
        {
            Disconnect();
        }
    }
}

// RobotPatrolSystemListener
// -----
public void debug(String msg) {
    System.out.println(msg);
}
public void isIdle(boolean idleFlag){
    idleFlag_ = idleFlag;

    if ( idleFlag_ ){
        btnStart.setEnabled(true);
        btnStop.setEnabled(false);
    }
    else {
        btnStart.setEnabled(false);
        btnStop.setEnabled(true);
    }
}

public void patrolStopped()
{
    btnStop.setEnabled(false);
    btnStart.setEnabled(true);
}

public static void main(String args[])
{
    UserInterfaceSubSystem dialog = new
UserInterfaceSubSystem();

```



```

        dialog.setVisible(true);
    }
}

+++++
package guiutil;

/**
 * RobotPatroSubSystem
 *
 */
public class Protocol
{
    public static final int PROTOCOL_DUMMY = 999;
    public static final int PROTOCOL_CLOSECONNECTION = 000;
    public static final int PROTOCOL_STARTPATROL = 100;
    public static final int PROTOCOL_STOPPATROL = 200;
    public static final int PROTOCOL_PATROLSTOPPED = 300;
    public static final int PROTOCOL_DEBUG = 400;
    public static final int PROTOCOL_RESULTOK = 800;
    public static final int PROTOCOL_RESULTERROR = 900;
    public static final int PROTOCOL_ISIDLE = 777;
}

+++++
package guiutil.rpsa;

import guiutil.Protocol;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.util.concurrent.ArrayBlockingQueue;

import lejos.pc.comm.NXTComm;
import lejos.pc.comm.NXTCommException;
import lejos.pc.comm.NXTCommFactory;
import lejos.pc.comm.NXTInfo;

/**
 * RobotPatrolSystem
 *
 */
public class RobotPatrolSystemAgent implements Runnable

```



```

{
    private NXTComm nxtComm = NXTCommFactory
        .createNXTComm(NXTCommFactory.BLUETOOTH);
    private DataInputStream dis;
    private DataOutputStream dos;
    private ArrayBlockingQueue queue = new
ArrayBlockingQueue(100);
    private RobotPatrolSystemListener listener = null;
    private boolean isRunning = false;

    public boolean connect(String name, String address)
    {
        boolean result = false;

        if (name == null || name.length() == 0 || address == null ||
address.length() == 0)
        {
            System.out.println("Name DANIMASTER, Address
00165302809a");
            System.exit(1);
        }

        NXTInfo[] nxtInfo = new NXTInfo[1];

        nxtInfo[0] = new NXTInfo(name, address);

        System.out.println("Connecting to " +
nxtInfo[0].btResourceString);

        try
        {
            result = nxtComm.open(nxtInfo[0]);
        } catch (NXTCommException e)
        {
            System.out.println("Exception from open");
        }

        System.out.println("Connected to " +
nxtInfo[0].btResourceString);
        result = true;

        dis = new DataInputStream(nxtComm.getInputStream());
        dos = new DataOutputStream(nxtComm.getOutputStream());

        isRunning = true;
        Thread t = new Thread(this);
        t.start();

        return result;
    }
}

```



```

public void close()
{
    try
    {
        isRunning = false;
        writeDummy();

        dos.writeInt(Protocol.PROTOCOL_CLOSECONNECTION);
        dos.flush();

        synchronized(this)
        {
            this.wait(1000);
        }

        dis.close();
        dos.close();
        nxtComm.close();
    }
    catch (Exception e)
    {
        System.out.println("IOException closing connection:");
        System.out.println(e.getMessage());
    }
}

public boolean startPatrol(int endPointX, int endPointY)
{
    boolean result = false;

    try
    {
        writeDummy();
        dos.writeInt(Protocol.PROTOCOL_STARTPATROL);
        dos.writeInt(endPointX);
        dos.writeInt(endPointY);
        dos.flush();

        int resultMsg = ((Integer) queue.take()).intValue();

        if (resultMsg == Protocol.PROTOCOL_RESULTOK)
        {
            result = true;
        } else if (resultMsg ==
Protocol.PROTOCOL_RESULTERROR)
        {
            result = false;
        }
    }
}

```



```

        } catch (Exception e)
        {
            e.printStackTrace();
        }

        return result;
    }

    public boolean stopPatrol()
    {
        boolean result = true;

        try
        {
            writeDummy();
            dos.writeInt(Protocol.PROTOCOL_STOPPATROL);
            dos.flush();

            int resultMsg = ((Integer) queue.take()).intValue();

            if (resultMsg == Protocol.PROTOCOL_RESULTOK)
            {
                result = true;
            } else if (resultMsg ==
Protocol.PROTOCOL_RESULTERROR)
            {
                result = false;
            }

        } catch (Exception e)
        {
            e.printStackTrace();
        }

        return result;
    }

    private void writeDummy() throws IOException
    {
        dos.writeInt(Protocol.PROTOCOL_DUMMY);
        dos.writeInt(Protocol.PROTOCOL_DUMMY);
        dos.writeInt(Protocol.PROTOCOL_DUMMY);
    }

    public void run()
    {
        while (isRunning)
        {
            int resultMsg;

```



```

        try
        {
            readDummy();
            resultMsg = dis.readInt();

            if (resultMsg ==
Protocol.PROTOCOL_RESULTOK)
            {
                queue.add(new Integer(resultMsg));
            } else if (resultMsg ==
Protocol.PROTOCOL_RESULTERROR)
            {
                queue.add(new Integer(resultMsg));
            }
            else if (resultMsg ==
Protocol.PROTOCOL_PATROLSTOPPED)
            {
                if(listener != null)
                {
                    listener.patrolStopped();
                }
                this.close();
            }
            else if (resultMsg ==
Protocol.PROTOCOL_DEBUG)
            {
                if(listener != null)
                {
                    int length = dis.readInt();
                    char[] msgChars = new
char[length];
                    for(int i=0; i<msgChars.length;
i++)
                    {
                        msgChars[i] =
dis.readChar();
                    }
                    listener.debug(new
String(msgChars));
                }
            }
            else if (resultMsg ==
Protocol.PROTOCOL_ISIDLE)
            {
                if(listener != null)
                {
                    boolean idleFlag =
dis.readBoolean();

```



```

                                listener.isIdle(idleFlag);
                                }
                            }
                        }
                    catch (EOFException e)
                    {
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                        break;
                    }
                }
            }

private void readDummy() throws IOException
{
    dis.readInt();
    dis.readInt();
    dis.readInt();
}

public RobotPatrolSystemListener getListener()
{
    return listener;
}

public void setListener(RobotPatrolSystemListener listener)
{
    this.listener = listener;
}
}

+++++
package guiutil.rpsa;

public interface RobotPatrolSystemListener
{
    public void patrolStopped();

    public void debug(String msg);

    public void isIdle(boolean idleFlag);
}

```



SECTION IV DATASHEETS



As this project is basically about programming, instead of attach datasheets of equipment, in this section the main functions of the API of Lego, which use the robot, will be shown, in order to a better understanding of the source code.

Ref [LEJOS]

6 Package lejos.navigation

Navigation classes.

See:

[Description](#)

Interface Summary

Navigator	The Navigator interface contains methods for performing basic navigational movements.
Pilot	The Pilot interface forms a common set of functions an implementation must offer in order to be used by higher level navigation classes.

Class Summary

CompassNavigator	The CompassNavigator class, like its superclass, can keep track of the robot position and the direction angle it faces.
----------------------------------	---



<u>CompassPilot</u>	A Pilot that keeps track of direction using a CompassSensor.
<u>SimpleNavigator</u>	The SimpleNavigator class can keep track of the robot position and the direction angle it faces; It uses a _pilot object to control NXT robot movements. The position and direction angle values are updated automatically when the movement command returns after the movement is complete and and after stop() command is issued.
<u>TachoNavigator</u>	Deprecated. <i>in 0.8, use SimpleNavigator</i>
<u>TachoPilot</u>	The TachoPilot class is a software abstraction of the Pilot mechanism of a NXT robot.
<u>WaypointNavigator</u>	The WaypointNavigator class extends the SimpleNavigator allowing the specify a path along a set of coordinates.

7 lejos.navigation

Interface Pilot

All Known Implementing Classes:

[CompassPilot](#), [TachoPilot](#)

```
public interface Pilot
```

The Pilot interface forms a common set of functions an implementation must offer in order to be used by higher level navigation classes. The Pilot hides the details of the robots physical construction and the required control algorithms from the rest of this package.



Method Summary

void	<u>backward()</u> Moves the NXT robot backward until stop() is called.
void	<u>forward()</u> Moves the NXT robot forward until stop() is called.
float	<u>getAngle()</u>
float	<u>getMoveMaxSpeed()</u>
float	<u>getMoveSpeed()</u>
float	<u>getTravelDistance()</u>
float	<u>getTurnMaxSpeed()</u>
float	<u>getTurnSpeed()</u>
boolean	<u>isMoving()</u>
void	<u>reset()</u> Reset traveled distance and rotated angle.
void	<u>rotate()</u> (float angle) Rotates the NXT robot a specific number of degrees in a direction.



void	<u>rotate</u> (float angle, boolean immediateReturn)	Rotates the NXT robot a specific number of degrees in a direction.
void	<u>setMoveSpeed</u> (float speed)	Sets the movement speed of the robot.
void	<u>setSpeed</u> (int speed)	Deprecated. in 0.8, use <i>setTurnSpeed()</i> and <i>setMoveSpeed()</i> . <i>The method was deprecated, as this it requires knowledge of the robots physical construction, which this interface should hide!</i>
void	<u>setTurnSpeed</u> (float speed)	Sets the turning speed of the robot.
void	<u>steer</u> (int turnRate)	Moves the NXT robot in a circular path at a specific turn rate.
void	<u>steer</u> (int turnRate, int angle)	Moves the NXT robot in a circular path at a specific turn rate.
void	<u>steer</u> (int turnRate, int angle, boolean immediateReturn)	Moves the NXT robot in a circular path at a specific turn rate.
void	<u>stop</u> ()	Halts the NXT robot and calculates new x, y coordinates.
void	<u>travel</u> (float distance)	Moves the NXT robot a specific distance.
void	<u>travel</u> (float distance, boolean immediateReturn)	Moves the NXT robot a specific distance.



void turn (float radius)	Moves the NXT robot in a circular path with a specified radius. The center of the turning circle is on the right side of the robot if parameter radius is negative. Postcondition: Motor speeds are unpredictable.
void turn (float radius, int angle)	Moves the NXT robot in a circular path with a specified radius. The center of the turning circle is on the right side of the robot if parameter radius is negative. Robot will stop when total rotation equals angle.
void turn (float radius, int angle, boolean immediateReturn)	Moves the NXT robot in a circular path with a specified radius. The center of the turning circle is on the right side of the robot if parameter radius is negative. Robot will stop when total rotation equals angle.

Method Detail

7.1 forward

void **forward**()

Moves the NXT robot forward until stop() is called.

See Also:

[Navigator.stop\(\)](#)

7.2 backward

void **backward**()

Daniel Egido Sánchez de Vega



Moves the NXT robot backward until stop() is called.

See Also:

[Navigator.stop\(\)](#)

7.3 stop

void **stop**()

Halts the NXT robot and calculates new x, y coordinates.

See Also:

[Navigator.forward\(\)](#)

7.4 isMoving

boolean **isMoving**()

Returns:

true if the robot is moving under power.

7.5 setMoveSpeed

void **setMoveSpeed**(float speed)

Sets the movement speed of the robot.

Parameters:

speed - The speed in wheel diameter units per second.

7.6 getMoveSpeed

float **getMoveSpeed**()

Returns:

the movement speed of the robot in wheel diameter units per second.



7.7 getMoveMaxSpeed

float **getMoveMaxSpeed**()

Returns:

the maximal movement speed of the robot in wheel diameter units per second which can be maintained accurately. Will change with time, as it is normally dependent on the battery voltage.

7.8 setTurnSpeed

void **setTurnSpeed**(float speed)

Sets the turning speed of the robot.

Parameters:

speed - The speed in degree per second.

7.9 getTurnSpeed

float **getTurnSpeed**()

Returns:

the turning speed of the robot in degree per second.

7.10 getTurnMaxSpeed

float **getTurnMaxSpeed**()

Returns:

the maximal turning speed of the robot in degree per second which can be maintained accurately. Will change with time, as it is normally dependent on the battery voltage.



7.11 setSpeed

void **setSpeed**(int speed)

Deprecated. *in 0.8, use setTurnSpeed() and setMoveSpeed(). The method was deprecated, as this it requires knowledge of the robots physical construction, which this interface should hide!*

Sets drive motor speed.

Parameters:

speed - The speed of the drive motor(s) in degree per second.

7.12 travel

void **travel**(float distance)

Moves the NXT robot a specific distance. A positive value moves it forward and a negative value moves it backward. Method returns when movement is done.

Parameters:

distance - The positive or negative distance to move the robot.

7.13 travel

void **travel**(float distance,
boolean immediateReturn)

Moves the NXT robot a specific distance. A positive value moves it forward and a negative value moves it backward.

Parameters:

distance - The positive or negative distance to move the robot.

immediateReturn - If immediateReturn is true then the method returns immediately and your code MUST call updatePostion() when the robot has stopped. Otherwise, the robot position is lost.



7.14 rotate

```
void rotate(float angle)
```

Rotates the NXT robot a specific number of degrees in a direction.

Method returns when rotation is done.

Parameters:

`angle` - The angle to rotate in degrees. A positive value rotates left, a negative value right (clockwise).

7.15 rotate

```
void rotate(float angle,  
            boolean immediateReturn)
```

Rotates the NXT robot a specific number of degrees in a direction.

Method returns when rotation is done.

Parameters:

`angle` - The angle to rotate in degrees. A positive value rotates left, a negative value right (clockwise).

`immediateReturn` - If `immediateReturn` is true then the method returns immediately and your code MUST call `updatePostion()` when the robot has stopped. Otherwise, the robot position is lost.

7.16 getAngle

```
float getAngle()
```

Returns:

the angle of rotation of the robot since last call to reset.

7.17 getTravelDistance

```
float getTravelDistance()
```

Returns:



distance traveled by the robot since last call to reset.

7.18 steer

```
void steer(int turnRate)
```

Moves the NXT robot in a circular path at a specific turn rate. The center of the turning circle is on the right side of the robot if parameter turnRate is negative. Values for turnRate are between -200 and +200. The turnRate determines the ratio of inner wheel speed to outer wheel speed (as a percent). *Formula:* ratio = 100 - abs(turnRate). When the ratio is negative, the outer and inner wheels rotate in opposite directions.

Examples:

- steer(25) -> inner wheel turns at 75% of the speed of the outer wheel
- steer(100) -> inner wheel stops
- steer(200) -> means that the inner wheel turns at the same speed as the outer wheel - a zero radius turn.

Note: Even if you have specified a drift correction in the constructor it will not be applied in this method.

Parameters:

turnRate - If positive, the left wheel is on the inside of the turn. If negative, the left wheel is on the outside.

7.19 steer

```
void steer(int turnRate,  
            int angle)
```

Moves the NXT robot in a circular path at a specific turn rate. The center of the turning circle is on the right side of the robot if parameter turnRate is negative. Values for turnRate are between -200 and +200. The turnRate determines the ratio of



inner wheel speed to outer wheel speed (as a percent).

Formula: $\text{ratio} = 100 - \text{abs}(\text{turnRate})$. When the ratio is negative, the outer and inner wheels rotate in opposite directions.

Examples:

- `steer(25)` -> inner wheel turns at 75% of the speed of the outer wheel
- `steer(100)` -> inner wheel stops
- `steer(200)` -> means that the inner wheel turns at the same speed as the outer wheel - a zero radius turn.

Note: Even if you have specified a drift correction in the constructor it will not be applied in this method.

Parameters:

`turnRate` - If positive, the left wheel is on the inside of the turn. If negative, the left wheel is on the outside.

`angle` - The angle through which the robot will rotate. If negative, robot traces the turning circle backwards.

7.20 steer

```
void steer(int turnRate,
            int angle,
            boolean immediateReturn)
```

Moves the NXT robot in a circular path at a specific turn rate. The center of the turning circle is on the right side of the robot if parameter `turnRate` is negative. Values for `turnRate` are between -200 and +200. The `turnRate` determines the ratio of inner wheel speed to outer wheel speed (as a percent).
Formula: $\text{ratio} = 100 - \text{abs}(\text{turnRate})$. When the ratio is negative, the outer and inner wheels rotate in opposite directions.

Examples:

- `steer(25)` -> inner wheel turns at 75% of the speed of the outer wheel



- `steer(100)` -> inner wheel stops
- `steer(200)` -> means that the inner wheel turns at the same speed as the outer wheel - a zero radius turn.

Note: Even if you have specified a drift correction in the constructor it will not be applied in this method.

Parameters:

`turnRate` - If positive, the left wheel is on the inside of the turn. If negative, the left wheel is on the outside.

`angle` - The angle through which the robot will rotate. If negative, robot traces the turning circle backwards.

`immediateReturn` - If `immediateReturn` is true then the method returns immediately and your code MUST call `updatePosition()` when the robot has stopped. Otherwise, the robot position is lost.

7.21 turn

void **turn**(float radius)

Moves the NXT robot in a circular path with a specified radius.

The center of the turning circle is on the right side of the robot if parameter radius is negative.

Postcondition: Motor speeds are unpredictable.

Parameters:

`radius` - of the circular path. If positive, the left wheel is on the inside of the turn. If negative, the left wheel is on the outside.

7.22 turn

void **turn**(float radius,
int angle)

Moves the NXT robot in a circular path with a specified radius.

The center of the turning circle is on the right side of the



robot if parameter radius is negative.
 Robot will stop when total rotation equals angle. If angle is negative, robot will move travel backwards.

Parameters:

radius - The radius of the turning circle.

angle - The sign of the angle determines the direction of robot motion.

7.23 turn

```
void turn(float radius,
           int angle,
           boolean immediateReturn)
```

Moves the NXT robot in a circular path with a specified radius. The center of the turning circle is on the right side of the robot if parameter radius is negative. Robot will stop when total rotation equals angle. If angle is negative, robot will move travel backwards.

Parameters:

radius - The radius of the turning circle.

angle - The sign of the angle determines the direction of robot motion.

immediateReturn - If immediateReturn is true then the method returns immediately and your code MUST call updatePostion() when the robot has stopped. Otherwise, the robot position is lost.

7.24 reset

```
void reset()
```

Reset traveled distance and rotated angle.



8 Package lejos.nxt

Access to NXT sensors, motors, etc.

See:

[Description](#)

Interface Summary

<u>ADSensorPort</u>	An abstraction for a port that supports Analog/Digital sensors.
<u>BasicMotorPort</u>	An abstraction for a motor port that supports RCX type motors, but not NXT motors with tachometers.
<u>BasicSensorPort</u>	An abstraction for a sensor port that supports setting and retrieving types and modes of sensors.
<u>ButtonListener</u>	Abstraction for receiver of button events.
<u>I2CPort</u>	Abstraction for a port that supports I2C sensors.
<u>LegacySensorPort</u>	Abstraction for a port that supports legacy RCX sensors.
<u>ListenerCaller</u>	Interface for calling calling lejos listeners.
<u>SensorConstants</u>	Constants used to set Sensor types and modes.
<u>SensorPortListener</u>	Interface for monitoring changes to the value for an Analogue/Digital sensor (such as a Touch, Light or Sound sensor) on a SensorPort.
<u>Tachometer</u>	Abstraction for the tachometer built into NXT motors.



<u>TachoMotorPort</u>	Abstraction for a motor port that supports NXT motors with tachometers.
---------------------------------------	---

Class Summary

<u>BasicMotor</u>	An abstraction for a motor without a tachometer, such as an RCX motor.
<u>Battery</u>	Provides access to Battery.
<u>Button</u>	Abstraction for an NXT button.
<u>Flash</u>	Read and write access to flash memory in pages.
<u>I2CSensor</u>	Abstract class that implements common methods for all I2C sensors.
<u>LCD</u>	Text and graphics output to the LCD display.
<u>LCDOutputStream</u>	A simple output stream that implements console output.
<u>LightSensor</u>	Abstraction for a NXT light sensor.
<u>Motor</u>	Abstraction for a NXT motor.
<u>MotorPort</u>	Abstraction for a NXT output port.
<u>NXT</u>	Abstraction for the local NXT Device.
<u>Poll</u>	Provides blocking access to events from the NXT.
<u>SensorPort</u>	Abstraction for a NXT input port.
<u>Settings</u>	leJOS NXJ persistent settings.



<u>Sound</u>	NXT sound routines.
<u>SoundSensor</u>	Abstraction for a NXT sound sensor.
<u>SystemSettings</u>	This class is designed for use by other lejos classes to read persistent settings.
<u>TextMenu</u>	Displays a list of items.
<u>TouchSensor</u>	Abstraction for a NXT touch sensor.
<u>UltrasonicSensor</u>	Abstraction for a NXT Ultrasonic Sensor.

Error Summary

[FlashError](#)

9 lejos.nxt

Class UltrasonicSensor

```
java.lang.Object
├── lejos.nxt.I2CSensor
│   └── lejos.nxt.UltrasonicSensor
```

All Implemented Interfaces:

[SensorConstants](#)

```
public class UltrasonicSensor
    extends I2CSensor
```

Abstraction for a NXT Ultrasonic Sensor.

WARNING: THIS CLASS IS SHARED BETWEEN THE classes AND pccomms PROJECTS. DO NOT EDIT THE VERSION IN pccomms AS IT WILL BE OVERWRITTEN WHEN THE PROJECT IS BUILT.



Field Summary

Fields inherited from interface `lejos.nxt.SensorConstants`

[`MODE\_ANGLESTEP`](#), [`MODE\_BOOLEAN`](#), [`MODE\_CELSIUS`](#), [`MODE\_FARENHEIT`](#),
[`MODE\_PCTFULLSCALE`](#), [`MODE\_PERIODCOUNTER`](#), [`MODE\_RAW`](#),
[`MODE\_TRANSITIONCNT`](#), [`TYPE\_ANGLE`](#), [`TYPE\_CUSTOM`](#), [`TYPE\_LIGHT\_ACTIVE`](#),
[`TYPE\_LIGHT\_INACTIVE`](#), [`TYPE\_LOWSPEED`](#), [`TYPE\_LOWSPEED\_9V`](#),
[`TYPE\_NO\_SENSOR`](#), [`TYPE\_REFLECTION`](#), [`TYPE\_SOUND\_DB`](#), [`TYPE\_SOUND\_DBA`](#),
[`TYPE\_SWITCH`](#), [`TYPE\_TEMPERATURE`](#)

Constructor Summary

[`UltrasonicSensor`](#)([`I2CPort`](#) port)

Method Summary

int	<code>capture</code> () Set capture mode Set the sensor into capture mode.
int	<code>continuous</code> () Switch to continuous ping mode.
int	<code>getCalibrationData</code> (byte[] data) Return 3 bytes of calibration data.
byte	<code>getContinuousInterval</code> () Return the interval used in continuous mode.
int	<code>getData</code> (int register, byte[] buf, int len) Executes an I2C read transaction and waits for the result.



int	<u>getDistance()</u> Return distance to an object.
int	<u>getDistances</u> (int[] dist) Return an array of 8 echo distances.
int	<u>getFactoryData</u> (byte[] data) Return 10 bytes of factory calibration data.
byte	<u>getMode</u> () Returns the current operating mode of the sensor.
<u>String</u>	<u>getUnits</u> () Return a string indicating the type of units in use by the unit.
int	<u>off</u> () Turn off the sensor.
int	<u>ping</u> () Send a single ping.
int	<u>reset</u> () Reset the device Performs a "soft reset" of the device.
int	<u>sendData</u> (int register, byte[] buf, int len) Executes an I2C write transaction.
int	<u>setCalibrationData</u> (byte[] data) Set 3 bytes of calibration data.
int	<u>setContinuousInterval</u> (byte interval) Set the ping inetrval used when in continuous mode.

Methods inherited from class `lejos.nxt.I2CSensor`

[fetchString](#), [getPort](#), [getProductID](#), [getSensorType](#), [getVersion](#),
[sendData](#), [setAddress](#)



Methods inherited from class java.lang.[Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#),
[toString](#), [wait](#), [wait](#), [wait](#)

Constructor Detail**9.1 UltrasonicSensor**

```
public UltrasonicSensor(I2CPort port)
```

Method Detail**9.2 getData**

```
public int getData(int register,  
                   byte[] buf,  
                   int len)
```

Description copied from class: [I2CSensor](#)

Executes an I2C read transaction and waits for the result.

Overrides:

[getData](#) in class [I2CSensor](#)

Parameters:

`register` - I2C register, e.g 0x41

`buf` - Buffer to return data

`len` - Length of the return data

Returns:

status == 0 success, != 0 failure

9.3 sendData

```
public int sendData(int register,  
                   byte[] buf,
```

Daniel Egido Sánchez de Vega



```
int len)
```

Description copied from class: [I2CSensor](#)

Executes an I2C write transaction.

Overrides:

[sendData](#) in class [I2CSensor](#)

Parameters:

`register` - I2C register, e.g 0x42

`buf` - Buffer containing data to send

`len` - Length of data to send

Returns:

status zero=success, non-zero=failure

9.4 getDistance

```
public int getDistance()
```

Return distance to an object. To ensure that the data returned is valid this method may have to wait a short while for the distance data to become available.

Returns:

distance or 255 if no object in range

9.5 getDistances

```
public int getDistances(int[] dist)
```

Return an array of 8 echo distances. These are generated when using ping mode. A value of 255 indicates that no echo was obtained. The array must contain at least 8 elements, if not -1 is returned. If the distance data is not yet available the method will wait until it is.



Returns:

0 if ok <> 0 otherwise

9.6 ping

```
public int ping()
```

Send a single ping. The sensor operates in two modes, continuous and ping. When in continuous mode the sensor sends out pings as often as it can and the most recently obtained result is available via a call to `getDistance`. When in ping mode a ping is only transmitted when a call is made to this method. This sends a single ping and up to 8 echoes are captured. These may be read by making a call to `getDistance` and passing a suitable array. A delay of approximately 20ms is required between the call to `ping` and obtaining the results. The `getDistance` call automatically takes care of this. The normal `getDistance` call may also be used with `ping`, returning information for the first echo. Calling this method will disable the default continuous mode, to switch back to continuous mode call `continuous`.

Returns:

0 if ok <> 0 otherwise

9.7 continuous

```
public int continuous()
```

Switch to continuous ping mode. This method enables continuous ping and capture mode. This is the default operating mode of the sensor. Please the notes for `ping` for more details.

Returns:

0 if ok <> 0 otherwise



9.8 off

```
public int off()
```

Turn off the sensor. This call disables the sensor. No pings will be issued after this call, until either ping, continuous or reset is called.

Returns:

0 if ok <> 0 otherwise

9.9 capture

```
public int capture()
```

Set capture mode Set the sensor into capture mode. The Lego documentation states: "Within this mode the sensor will measure whether any other ultrasonic sensors are within the vicinity. With this information a program can evaluate when it is best to make a new measurement which will not conflict with other ultrasonic sensors." I have no way of testing this. Perhaps someone with a second NXT could check it out!

Returns:

0 if ok <> 0 otherwise

9.10 reset

```
public int reset()
```

Reset the device Performs a "soft reset" of the device. Restores things to the default state. Following this call the sensor will be operating in continuous mode.

Returns:

0 if ok <> 0 otherwise



9.11 getFactoryData

public int **getFactoryData**(byte[] data)

Return 10 bytes of factory calibration data. The bytes are as follows

data[0] : Factory zero (cal1) data[1] : Factory scale factor (cal2)

data[2] : Factory scale divisor.

Returns:

0 if ok <> 0 otherwise

9.12 getUnits

public [String](#) **getUnits**()

Return a string indicating the type of units in use by the unit. The

default response is 10E-2m indicating centimetres in use.

Returns:

7 byte string

9.13 getCalibrationData

public int **getCalibrationData**(byte[] data)

Return 3 bytes of calibration data. The bytes are as follows data[0] :

zero (cal1) data[1] : scale factor (cal2) data[2] : scale divisor.

Returns:

0 if ok <> 0 otherwise

9.14 setCalibrationData

public int **setCalibrationData**(byte[] data)

Set 3 bytes of calibration data. The bytes are as follows data[0] : zero



(cal1) data[1] : scale factor (cal2) data[2] : scale divisor. This does not currently seem to work.

Returns:

0 if ok <> 0 otherwise

9.15 getContinuousInterval

public byte **getContinuousInterval**()

Return the interval used in continuous mode. This seems to be in the range 1-15. It can be read and set. However tests seem to show it has no effect. Others have reported that this does vary the ping interval (when used in other implementations). Please report any new results.

Returns:

-1 if error otherwise the interval

9.16 setContinuousInterval

public int **setContinuousInterval**(byte interval)

Set the ping interval used when in continuous mode. See getContinuousInterval for more details.

Returns:

0 if ok <> 0 otherwise.

9.17 getMode

public byte **getMode**()

Returns the current operating mode of the sensor. 0 : sensor is off 1 : Single shot ping mode 2 : continuous ping mode (default) 3 : Event capture mode



Returns:

-1 if error otherwise the operating mode

10 lejos.nxt

Class LightSensor

[java.lang.Object](#)
└─ [lejos.nxt.LightSensor](#)

All Implemented Interfaces:

[SensorConstants](#)

```
public class LightSensor
  extends Object
  implements SensorConstants
```

Abstraction for a NXT light sensor. The light sensor can be calibrated to low and high values.

WARNING: THIS CLASS IS SHARED BETWEEN THE classes AND pccomms PROJECTS. DO NOT EDIT THE VERSION IN pccomms AS IT WILL BE OVERWRITTEN WHEN THE PROJECT IS BUILT.

Field Summary

Fields inherited from interface [lejos.nxt.SensorConstants](#)

[MODE_ANGLESTEP](#), [MODE_BOOLEAN](#), [MODE_CELSIUS](#), [MODE_FARENHEIT](#),
[MODE_PCTFULLSCALE](#), [MODE_PERIODCOUNTER](#), [MODE_RAW](#),
[MODE_TRANSITIONCNT](#), [TYPE_ANGLE](#), [TYPE_CUSTOM](#), [TYPE_LIGHT_ACTIVE](#),
[TYPE_LIGHT_INACTIVE](#), [TYPE_LOWSPEED](#), [TYPE_LOWSPEED_9V](#),
[TYPE_NO_SENSOR](#), [TYPE_REFLECTION](#), [TYPE_SOUND_DB](#), [TYPE_SOUND_DBA](#),
[TYPE_SWITCH](#), [TYPE_TEMPERATURE](#)

Constructor Summary



[LightSensor](#)([ADSensorPort](#) port)

Create a light sensor object attached to the specified port.

[LightSensor](#)([ADSensorPort](#) port, boolean floodlight)

Create a light sensor object attached to the specified port, and sets floodlighting on or off.

Method Summary

void	<u>calibrateHigh</u> ()	call this method whtn the light sensor is reading the high value - used by reaeValue
void	<u>calibrateLow</u> ()	call this method when the light sensor is reading the low value - used by readValue
int	<u>getHigh</u> ()	return the normalized value corresponding to readValue() = 100;
int	<u>getLow</u> ()	return the normalized value corresponding to readValue() = 0
int	<u>readNormalizedValue</u> ()	Read the current sensor normalized value.
int	<u>readValue</u> ()	Read the current sensor value.
void	<u>setFloodlight</u> (boolean floodlight)	Set floodlighting on or off.
void	<u>setHigh</u> (int high)	set the normalized value corresponding to readValue() = 100;



void	<u>setLow</u> (int low)
------	---

set the normalized value corresponding to readValue() = 0

Methods inherited from class java.lang.[Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

Constructor Detail

10.1 LightSensor

```
public LightSensor(ADSensorPort port)
```

Create a light sensor object attached to the specified port. The sensor will be set to floodlit mode, i.e. the LED will be turned on.

Parameters:

port - port, e.g. Port.S1

10.2 LightSensor

```
public LightSensor(ADSensorPort port,  
                   boolean floodlight)
```

Create a light sensor object attached to the specified port, and sets floodlighting on or off.

Parameters:

port - port, e.g. Port.S1

floodlight - true to set floodlit mode, false for ambient light.

Method Detail

10.3 setFloodlight

```
public void setFloodlight(boolean floodlight)
```

Daniel Egido Sánchez de Vega



Set floodlighting on or off.

Parameters:

`floodlight` - true to set floodit mode, false for ambient light.

10.4 readValue

```
public int readValue()
```

Read the current sensor value. Use `calibrateLow()` to set the zero level, and `calibrateHigh` to set the 100 level.

Returns:

Value as a percentage of difference between the low and high calibration values.

10.5 readNormalizedValue

```
public int readNormalizedValue()
```

Read the current sensor normalized value. Allows more accuracy than `readValue()`. For LEGO sensor, values typically range from 145 (dark) to 890 (sunlight).

Returns:

Value as raw normalized (0 to 1023)

10.6 calibrateLow

```
public void calibrateLow()
```

call this method when the light sensor is reading the low value - used by `readValue`

10.7 calibrateHigh

```
public void calibrateHigh()
```

Daniel Egido Sánchez de Vega



call this method whtn the light sensor is reading the high value -
used by reaeValue

10.8 setLow

```
public void setLow(int low)
```

set the normalized value corresponding to readValue() = 0

Parameters:

low - the low value

10.9 setHigh

```
public void setHigh(int high)
```

set the normalized value corresponding to readValue() = 100;

Parameters:

high - the high value

10.10 getLow

```
public int getLow()
```

return the normalized value corresponding to readValue() = 0

10.11 getHigh

```
public int getHigh()
```

return the normalized value corresponding to readValue() = 100;

11 lejos.nxt

Class Motor

Daniel Egido Sánchez de Vega



```

java.lang.Object
└─ lejos.nxt.BasicMotor
   └─ lejos.nxt.Motor

```

```

public class Motor
extends BasicMotor

```

Abstraction for a NXT motor. Three instances of `Motor` are available: `Motor.A`, `Motor.B` and `Motor.C`. To control each motor use methods `forward`, `backward`, `reverseDirection`, `stop` and `flt`. To set each motor's speed, use `setSpeed`. Speed is in degrees per second. \ Methods that use the tachometer:

`regulateSpeed`, `rotate`, `rotateTo`. These rotate methods may not stop smoothly at the target angle if called when the motor is already moving

`Motor` has 2 modes : `speedRegulation` and `smoothAcceleration` which only works if speed regulation is used. These are initially enabled. The speed is regulated by comparing the tacho count with speed times elapsed time and adjusting motor power to keep these closely matched. Smooth acceleration corrects the speed regulation to account for the acceleration time. They can be switched off/on by the methods `regulateSpeed()` and `smoothAcceleration()`. The actual maximum speed of the motor depends on battery voltage and load. With no load, the maximum is about 100 times the voltage. Speed regulation fails if the target speed exceeds the capability of the motor. If you need the motor to hold its position and you find that still moves after `stop()` is called , you can use the `lock()` method.

Example:

```

Motor.A.setSpeed(720); // 2 RPM
Motor.C.setSpeed(720);
Motor.A.forward();
Motor.C.forward();
Thread.sleep (1000);
Motor.A.stop();
Motor.C.stop();
Motor.A.regulateSpeed(true);
Motor.A.rotateTo( 360);
Motor.A.rotate(-720,true);
while(Motor.A.isRotating());
int angle = Motor.A.getTachoCount(); // should be -360

```

Author:

Roger Glassey revised 9 Feb 2008 - added `lock()` method.

Nested Class Summary



class	<u>Motor.Regulator</u> inner class to regulate speed; also stop motor at desired rotation angle
-------	--

Field Summary

<u>TachoMotorPort</u>	<u>port</u>
boolean	<u>rampUp</u> set by smoothAcceleration, forward(),backward(), setSpeed().
static <u>Motor</u>	<u>A</u> Motor A.
static <u>Motor</u>	<u>B</u> Motor B.
static <u>Motor</u>	<u>C</u> Motor C.
<u>Motor.Regulator</u>	<u>regulator</u>

Constructor Summary

[Motor](#)([TachoMotorPort](#) port)

Use this constructor to assign a variable of type motor connected to a particular port.

Method Summary



void	<u>backward()</u> Causes motor to rotate backwards.
void	<u>flt()</u> Causes motor to float.
void	<u>forward()</u> Causes motor to rotate forward.
int	<u>getActualSpeed()</u> returns actualSpeed degrees per second, calculated every 100 ms; negative value means motor is rotating backward
float	<u>getBasePower()</u> for debugging
float	<u>getError()</u> for debugging
int	<u>getLimitAngle()</u> Return the angle that a Motor is rotating to.
int	<u>getMode()</u> Returns the mode.
int	<u>getPower()</u> Returns the current power setting.
int	<u>getSpeed()</u> Returns the current motor speed in degrees per second
int	<u>getStopAngle()</u>
int	<u>getTachoCount()</u> Returns the tachometer count.
boolean	<u>isMoving()</u> Returns true iff the motor is in motion.



boolean	<u>isRegulating()</u>	
boolean	<u>isRotating()</u>	returns true when motor rotation task is not yet complete a specified angle
void	<u>lock</u> (int power)	Applies power to hold motor in current position.
void	<u>regulateSpeed</u> (boolean yes)	turns speed regulation on/off; Cumulative speed error is within about 1 degree after initial acceleration.
void	<u>resetTachoCount</u> ()	Resets the tachometer count to zero.
void	<u>reverseDirection</u> ()	Reverses direction of the motor.
void	<u>rotate</u> (int angle)	causes motor to rotate through angle.
void	<u>rotate</u> (int angle, boolean immediateReturn)	causes motor to rotate through angle; iff immediateReturn is true, method returns immediately and the motor stops by itself If any motor method is called before the limit is reached, the rotation is canceled.
void	<u>rotateTo</u> (int limitAngle)	causes motor to rotate to limitAngle; Then getTachoCount should be within +- 2 degrees of the limit angle when the method returns



void	<u>rotateTo</u> (int limitAngle, boolean immediateReturn) causes motor to rotate to limitAngle; if immediateReturn is true, method returns immediately and the motor stops by itself and getTachoCount should be within +- 2 degrees if the limit angle If any motor method is called before the limit is reached, the rotation is canceled.
void	<u>setBrakePower</u> (int pwr)
void	<u>setPower</u> (int power) sets motor power.
void	<u>setSpeed</u> (int speed) Sets motor speed , in degrees per second; Up to 900 is possible with 8 volts.
void	<u>shutdown</u> () causes run() to exit
void	<u>smoothAcceleration</u> (boolean yes) enables smoother acceleration.
void	<u>stop</u> () Causes motor to stop, pretty much instantaneously.

Methods inherited from class lejos.nxt.[BasicMotor](#)

[isBackward](#), [isFloating](#), [isForward](#), [isStopped](#)

Methods inherited from class java.lang.[Object](#)

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)



Field Detail

11.1 _port

```
public TachoMotorPort _port
```

11.2 regulator

```
public Motor.Regulator regulator
```

11.3 _rampUp

```
public boolean _rampUp
```

set by smoothAcceleration, forward(),backward(), setSpeed(). Only has effect if _regulate is true

11.4 A

```
public static final Motor A
```

Motor A.

11.5 B

```
public static final Motor B
```

Motor B.

11.6 C

```
public static final Motor C
```

Motor C.

Constructor Detail



11.7 Motor

```
public Motor(TachoMotorPort port)
```

Use this constructor to assign a variable of type motor connected to a particular port.

Parameters:

`port` - to which this motor is connected

Method Detail

11.8 getStopAngle

```
public int getStopAngle()
```

11.9 forward

```
public void forward()
```

Causes motor to rotate forward.

Overrides:

[forward](#) in class [BasicMotor](#)

11.10 backward

```
public void backward()
```

Causes motor to rotate backwards.

Overrides:

[backward](#) in class [BasicMotor](#)

11.11 reverseDirection

```
public void reverseDirection()
```

Reverses direction of the motor. It only has effect if the motor is moving.



Overrides:

[reverseDirection](#) in class [BasicMotor](#)

11.12 flt

```
public void flt()
```

Causes motor to float. The motor will lose all power, but this is not the same as stopping. Use this method if you don't want your robot to trip in abrupt turns.

Overrides:

[flt](#) in class [BasicMotor](#)

11.13 stop

```
public void stop()
```

Causes motor to stop, pretty much instantaneously. In other words, the motor doesn't just stop; it will resist any further motion. Cancels any rotate() orders in progress

Overrides:

[stop](#) in class [BasicMotor](#)

11.14 lock

```
public void lock(int power)
```

Applies power to hold motor in current position. Use if stop() is not good enough to hold the motor in position against a load.

Parameters:

power -- a value between 1 and 100;



11.15 isMoving

```
public boolean isMoving()
```

Description copied from class: [BasicMotor](#)

Returns true iff the motor is in motion.

Overrides:

[isMoving](#) in class [BasicMotor](#)

Returns:

true iff the motor is currently in motion.

11.16 rotate

```
public void rotate(int angle)
```

causes motor to rotate through angle.

Parameters:

angle - through which the motor will rotate

11.17 rotate

```
public void rotate(int angle,  
                   boolean immediateReturn)
```

causes motor to rotate through angle;

iff immediateReturn is true, method returns immediately and the

motor stops by itself

If any motor method is called before the limit is reached, the rotation is canceled. When the angle is reached, the method `isRotating()` returns false;

Parameters:

angle - through which the motor will rotate

immediateReturn - iff true, method returns immediately, thus allowing monitoring of sensors in the calling thread.



11.18 rotateTo

```
public void rotateTo(int limitAngle)
```

causes motor to rotate to limitAngle;

Then getTachoCount should be within +- 2 degrees of the limit angle when the method returns

Parameters:

limitAngle - to which the motor will rotate

11.19 rotateTo

```
public void rotateTo(int limitAngle,  
                     boolean immediateReturn)
```

causes motor to rotate to limitAngle;

if immediateReturn is true, method returns immediately and the motor stops by itself

and getTachoCount should be within +- 2 degrees if the limit angle

If any motor method is called before the limit is reached, the rotation is canceled. When the angle is reached, the method isRotating() returns false;

Parameters:

limitAngle - to which the motor will rotate, and then stop.

immediateReturn - iff true, method returns immediately, thus allowing monitoring of sensors in the calling thread.

11.20 shutdown

```
public void shutdown()
```

causes run() to exit



11.21 regulateSpeed

public void **regulateSpeed**(boolean yes)

turns speed regulation on/off;

Cumulative speed error is within about 1 degree after initial acceleration.

Parameters:

yes - is true for speed regulation on

11.22 smoothAcceleration

public void **smoothAcceleration**(boolean yes)

enables smoother acceleration. Motor speed increases gently, and does not <> overshoot when regulate Speed is used.

11.23 setSpeed

public void **setSpeed**(int speed)

Sets motor speed , in degrees per second; Up to 900 is possible with 8 volts.

Parameters:

speed - value in degrees/sec

11.24 setPower

public void **setPower**(int power)

sets motor power. This method is used by the Regulator thread to control motor speed. Warning: negative power will cause the motor to run in reverse but without updating the _direction field which is used by the Regulator thread. If the speed regulation is enabled, the results are unpredictable.

Overrides:



[setPower](#) in class [BasicMotor](#)

Parameters:

power - power setting: 0 - 100

11.25 getSpeed

```
public int getSpeed()
```

Returns the current motor speed in degrees per second

11.26 getMode

```
public int getMode()
```

Description copied from class: [BasicMotor](#)

Returns the mode.

Overrides:

[getMode](#) in class [BasicMotor](#)

Returns:

: 1 = forward, 2= backward, 3 = stop, 4 = float

11.27 getPower

```
public int getPower()
```

Description copied from class: [BasicMotor](#)

Returns the current power setting.

Overrides:

[getPower](#) in class [BasicMotor](#)

Returns:

power value 0-100



11.28 getLimitAngle

```
public int getLimitAngle()
```

Return the angle that a Motor is rotating to.

Returns:

angle in degrees

11.29 isRotating

```
public boolean isRotating()
```

returns true when motor rotation task is not yet complete a specified angle

11.30 isRegulating

```
public boolean isRegulating()
```

11.31 getActualSpeed

```
public int getActualSpeed()
```

returns actualSpeed degrees per second, calculated every 100 ms;
negative value means motor is rotating backward

11.32 getTachoCount

```
public int getTachoCount()
```

Returns the tachometer count.

Returns:

tachometer count in degrees



11.33 resetTachoCount

```
public void resetTachoCount()
```

Resets the tachometer count to zero.

11.34 getError

```
public float getError()
```

for debugging

Returns:

regulator error

11.35 getBasePower

```
public float getBasePower()
```

for debugging

Returns:

base power of regulator

11.36 setBrakePower

```
public void setBrakePower(int pwr)
```

